

Traffic Simulation using Agent-Based Modelling

Andrew Lansdowne
02973407

BSc (Hons) Computer Science

University of the West of England

2006

Abstract

This report follows the design and implementation of a traffic simulator written in NetLogo, an agent-based modelling environment. The system is developed with a view to analysing the likely effect of congestion-reducing schemes, in particular high occupancy vehicle lanes. The simulator is calibrated and validated using data from traffic studies, and then a section of road in South Gloucestershire is modelled to investigate the introduction of high occupancy vehicle lanes. The results suggest that the high occupancy vehicle lane fulfils its objectives by significantly reducing travel times for car sharers without adversely affecting other drivers.

Acknowledgments

I would like to thank my project supervisor Chris Wallace for his knowledge and guidance throughout the project. Thank you to my fiancée Megan Perry for keeping me on track with her loving support. Thanks to my good friend Dan Clark for his helpful suggestions, and to the A-Team for providing the daily entertainment.

Table of Contents

Abstract	i
Acknowledgments	ii
Table of Contents	iii
List of Tables.....	vi
List of Figures	vii
Chapter 1: Introduction	1
Chapter 2: Traffic Design & Analysis	2
2.1 Traffic Congestion.....	2
2.2 Tackling Traffic Congestion.....	3
2.2.1 Existing Schemes.....	3
2.2.2 High Occupancy Vehicle Lanes	4
2.2.3 Local Traffic Study.....	5
2.3 Traffic Theory.....	6
2.3.1 Measuring Traffic Flow	6
2.3.2 Modelling Traffic Flow.....	6
2.3.3 Understanding Drivers	9
Chapter 3: Traffic Simulation.....	10
3.1 Introduction to Traffic Simulation	10
3.2 Simulation Classifications.....	10
3.3 Macroscopic vs. Microscopic	11
3.4 Simulation Scenarios	12
3.5 Modelling Issues	13
3.5.1 Modelling Vehicles	13
3.5.2 Modelling Drivers.....	13
3.5.3 Modelling the Environment	14
3.6 Simulation Issues	15
3.6.1 Vehicle Following.....	15
3.6.2 Lane Changing	16
3.6.3 Urban Simulation Issues.....	17
3.6.4 Accuracy and Time Issues.....	17
3.7 Requirements of Traffic Simulators	18
Chapter 4: Agent-Based Traffic Simulation	22
4.1 Multi-Agent Systems	22
4.1.1 Agents.....	22

4.1.2	Agent Environments.....	22
4.1.3	Differences to Object-Oriented Programming.....	23
4.2	Agent-Based Modelling (ABM).....	24
4.2.1	Multi-Agent Modelling and Simulation.....	24
4.2.2	Calibration and Validation	24
4.2.3	System Development	25
4.2.4	Integrated Development Environments.....	26
4.3	Multi-Agent Traffic Simulations	30
Chapter 5: Design.....		31
5.1	Project Design.....	31
5.1.1	Project Overview	31
5.1.2	Software Development Process Model.....	31
5.1.3	User Analysis & Use Cases	32
5.1.4	User Requirements	34
5.1.5	Choice of Tool.....	35
5.2	Software Design.....	36
5.2.1	System Requirements.....	36
5.2.2	Using NetLogo	38
5.2.3	Class Diagram	40
5.2.4	Interaction Diagram	41
5.3	Test Plan	42
5.3.1	Error Testing.....	42
5.3.2	Microscopic Validation.....	42
5.3.3	Macroscopic Validation.....	43
Chapter 6: Implementation and Evaluation.....		44
6.1	Phase One.....	44
6.1.1	Phase Implementation Overview.....	44
6.1.2	Phase Implementation Details and Issues	45
6.1.3	Phase Testing	45
6.2	Phase Two	46
6.2.1	Phase Implementation Overview.....	46
6.2.2	Phase Implementation Details	47
6.2.3	Phase Testing	48
6.3	Phase Three	49
6.3.1	Phase Implementation Overview.....	49
6.3.2	Phase Implementation Details	49
6.3.3	Phase Testing	49

6.4	Testing on the Problem.....	50
6.4.1	Modelling the Network	50
6.4.2	Simulating the Network.....	53
6.4.3	Interpreting the Results	55
6.4.4	Performance Testing.....	55
Chapter 7: Conclusions and Future Work.....		57
Bibliography		59
Appendix A: South Gloucestershire HOV Lanes.....		63
A-1	Statistical Data Received.....	63
A-2	Map Data Received	65
Appendix B: Additional Implementation Details		66
B-1	Calibrating the Vehicle-following Model.....	66
B-2	Calibrating Vehicle Parameters	67
B-3	Example Time-Space Charts	67
B-4	Example Flow-Density Chart.....	70
Appendix C: Simulation Code.....		71
Appendix D: Graphical Interface		83

Note: the supporting website for this project contains links, code and runnable applets.

It is available at <http://www.myhomezone.co.uk/project/>

List of Tables

Table 3.1: Selection of Traffic Simulators by Type.....	19
Table 5.1: Microscopic Validation Test Plan	43
Table 6.1: Requirements Implemented in Phase One	44
Table 6.2: Requirements Implemented in Phase Two.....	47
Table 6.3: Requirements Implemented in Phase Three	49

List of Figures

Figure 2.1: Estimated UK Road Traffic Volume.	2
Figure 2.2: UWE Local Traffic Area.	5
Figure 2.3: Stylised traffic flow-density diagram.....	7
Figure 2.4: Space-time diagram where lines represent vehicles, observed using aerial photography.	8
Figure 2.5: Flow-density diagram incorporating synchronised flow.	9
Figure 3.1: Observation-decision diagram showing limits in different of speed (DV) and distance (DX) for selecting vehicle states.	16
Figure 3.2: Basic notation for lane change manoeuvres.....	16
Figure 3.3: Importance of Traffic Simulator Features.....	19
Figure 3.4: Importance of Simulator Indicators.	20
Figure 3.5: Transport Telematics and Technological Functions.	21
Figure 4.1: Evolution of Agent Programming.....	24
Figure 4.2: NetLogo Interface.....	27
Figure 4.3: Breve Interface.....	28
Figure 4.4: SB-MASE Interface	29
Figure 4.5: SeSAm Interface	30
Figure 5.1: Use Case Diagram	33
Figure 5.2: System Class Diagram.....	40
Figure 5.3: Sequence Diagram of "go" procedure.....	41
Figure 6.1: Phase One Microscopic Tests T-1, T-2 and T-3.....	46
Figure 6.2: Phase One Flow-density Test.....	46
Figure 6.3: Phase Two Flow-density Test	48
Figure 6.4: Phase Two Microscopic Tests T-4 and T-5	49
Figure 6.5: Road to Model. Source: South Gloucestershire Council.	50
Figure 6.6: NetLogo World & View Screenshot	51
Figure 6.7: Model Editing Tools Screenshot	51
Figure 6.8: Model Creation Screenshot.....	52
Figure 6.9: User Input Screenshots	52
Figure 6.10: 3D Visualisation Screenshot.....	53
Figure 6.11: Model Overview Screenshot	53
Figure 6.12: Journey Times – Normal vs. HOV	54
Figure 6.13: Flow-Density Results.....	54
Figure 6.14: AVO Results	55

Chapter 1: Introduction

This project focuses on the use of traffic simulation in helping traffic engineers reduce congestion, in particular by use of high occupancy vehicle (HOV) lanes. Traffic congestion is a major problem, and with the volume of traffic increasing at a steady rate, it is set to get worse. There are many schemes for reducing congestion; one is the use of HOV lanes, which are now being introduced on congested roads in the UK and are used often in the USA. Traffic simulation can help to determine the likely effects of introducing various schemes without costly infrastructure changes.

Traffic simulation can be implemented using agent-based modelling (ABM), which enables dynamic objects such as vehicles to be modelled individually and have control over their behaviour. NetLogo is chosen as the ABM environment to be used in the project, and is used to develop a traffic simulator, which is then calibrated and verified with data from traffic research. The simulator is used to model part of a local road network with and without a HOV lane, and results are compared to claims of success from the local council.

The purpose of the project is to show that traffic simulation is a worthy tool for traffic engineers evaluating alternative schemes. It aims to develop a unique traffic simulation system that can be used to study traffic theory and assess network infrastructure and control changes. Main objectives include gaining an understanding of traffic theory, learning the major features and issues of traffic simulation, and evaluating agent-based modelling as a means of simulating traffic.

This report is structured as follows: Chapter 2 examines the problem of traffic congestion, various schemes for reducing congestion, and traffic flow theory. Chapter 3 looks in detail at the design features of traffic simulation systems, and Chapter 4 introduces agent-based modelling along with its application to traffic. Chapter 5 follows the design of the traffic simulator, and Chapter 6 follows its implementation in prototypical stages with details of results from testing on the problem. Project conclusions are presented in Chapter 7.

Chapter 2: Traffic Design & Analysis

2.1 Traffic Congestion

Traffic congestion has been a major problem on roads around the world for many years. Roman history indicates that in the first century, the streets of Rome became so congested that all non-official vehicles were banned from entering the city [39]. In modern cities, traffic on major roads is abundant, and steps have to be taken to keep the traffic flowing at an acceptable speed.

The volume of road traffic has increased rapidly in recent years. In the UK, total road traffic has almost doubled since 1980 (see Figure 2.1). Forecasts from the Department for Transport show that the volume of road traffic will continue to increase at an alarming rate. These forecasts, which in the past have been conservative estimates, suggest that traffic levels will increase by approximately 50% between the years 2000 and 2020 [43]. If this is the case, then the causes and effects of traffic congestion need to be understood now or it could become much worse a problem in the near future.

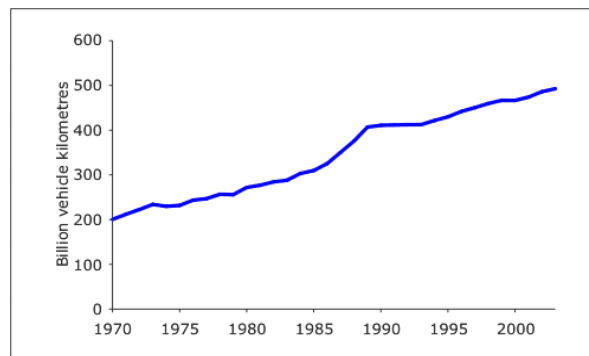


Figure 2.1: Estimated UK Road Traffic Volume. Source: [55]

At its most basic, congestion is caused when the volume of traffic exceeds road capacity. This holds for most perceived causes of congestion; for example, accidents, breakdowns and road works decrease the available road capacity, while school-run and holiday traffic increase the volume of traffic [50]. The Government has set targets for reducing congestion in its “10 Year Plan for Transport” [51]. This plan was published in the year 2000, and it outlines the Government’s goal of improving transport by the year 2010. In the plan, a simple definition of congestion has been adopted, which makes it quantifiable:

Congestion in this Plan is defined as the average delay experienced for each kilometre travelled compared to driving at speeds typical when traffic is light.

[51]

Using this definition, the Department for Transport has attempted to measure and forecast congestion levels by conducting studies on the existing road network. Without the effects of the plan, congestion was predicted to rise by 15% across the whole road network. The Government deemed this an unacceptable rise, and the plan sets the target to reduce congestion below current levels by 2010, particularly in large urban areas. Various strategies are outlined in the plan to help improve congestion; these are detailed in section 2.2. Critics argue that so far, congestion has risen since the start of the plan; however, the Government's response was that this is expected because congestion trends take time to reverse. [51], [52]

Traffic congestion is a major issue for many drivers; it results in journey delays, wasted time, increased pressure and can make people late or cause loss of business [50]. Congestion also adversely affects passengers, pedestrians, cyclists, and users of buses and taxis, causing more delay, stress and even danger. The environment and local residents may be affected by emissions or noise pollution associated to vehicles. Congestion reduces the quality of life for many people and deserves to be tackled to improve transport for everyone. [39]

2.2 Tackling Traffic Congestion

Many schemes have been designed and implemented with relieving congestion as a primary objective. These schemes have varying effectiveness, and it is not always obvious which measures will work best in a given situation. The most common methods are considered in section 2.2.1, with an indication of their effectiveness. A new scheme receiving a good deal of attention at present is the introduction of high occupancy vehicle lanes; this is discussed in section 2.2.2.

2.2.1 Existing Schemes

The immediately obvious strategy is to increase road capacity by building new roads or improving existing roads, but this rarely works to reduce long-term congestion. Congestion is self-managing, because traffic is governed by what is regarded as a tolerable level of congestion; as road capacity increases, demand (traffic) increases to fill the new capacity. This concept is known as Braess' Paradox or Pigou-Knight analysis, and suggests that increasing the network capacity can often be not only ineffective, but also counterproductive. This is also one of the most expensive ways of tackling congestion. [37], [25]

Improving public transport is an accepted way to improve overall transport in busy congested areas such as town centres. This covers bus lanes, improving other modes of transport and increasing awareness of the benefits of walking and cycling. It is one of the aims of the Government's plan for transport, and it is thought that if people feel there are real alternatives to driving, then travelling will be easier for everyone. [39], [51].

An attractive remedial measure for reducing congestion is Intelligent Transport Systems (ITS), which is concerned with the application of electronic information systems to control, manage and improve transport. This includes real-time route guidance systems, variable speed limits controlled by computer systems, real-time transport information (e.g. congestion levels, delay expected for public transport, variable message signs), and dynamic signal timings. [36], [43]. ITS makes possible many new traffic management opportunities, which help to reduce congestion by making the best use of the existing transport network. [8]

Congestion charging, a form of road pricing implemented in London, is a way of reducing demand (traffic) by forcing drivers to pay a fee when driving through a congested area such as a city centre. This form of restraint is designed to encourage drivers to use other forms of transport when travelling in an extremely congested area, and is very effective at reducing or eliminating congestion as well as increasing air quality and road safety, more predictable travel, and possibly profits. However, drivers often see it as an unfair money making scheme, congestion can be increased around the edge of the charging zone as drivers avoid it, more costs for businesses travelling within the zone, and reduced profits for businesses just inside the zone. [51], [53]

2.2.2 High Occupancy Vehicle Lanes

One attempt to improve efficiency of congested roads is to implement one or more high occupancy vehicle (HOV) lanes. These lanes are only to be used by vehicles carrying more than a certain number of passengers. Similar to bus lanes, they can be used by buses, taxis, but also by any other vehicle making efficient use of the road by carrying more passengers. HOV lanes are intended to promote the use of public transport and car sharing (car pools), to ease congestion by reducing the number of cars on the road, and to improve efficiency by increasing the throughput of vehicles carrying more people. [54]

HOV lanes appear to be a state-of-the-art technique for tackling traffic congestion in the United Kingdom. Only a few schemes have been implemented to date, and have involved dual carriageways in Leeds and South Gloucestershire. However, there are now proposals in place to convert one lane on motorways into a HOV lane during peak periods, on sections of the M1, M3, M61 and M62. There is a fair amount of controversy on HOV lanes, as many drivers only perceive larger amounts of congestion because of them, especially for single occupant vehicles. The USA has been at the forefront of using HOV lanes, and has common standards are in place for their planning and design. The majority of schemes involve restricting use of the HOV lane to vehicles with two or more occupants. [54]

The measure used to evaluate the performance of HOV lanes is the average vehicle occupancy (AVO) of a road. In the USA, the primary objective is to increase the AVO, regardless of resulting traffic flows. It is generally accepted that the introduction of a HOV lane will increase the AVO by 10-15%, however the resulting volume of vehicles is not so readily available. Depending on local conditions, the introduction of a HOV lane is expected to reduce the number of vehicles by approximately 10% through increased occupancy and some vehicles taking different routes. [54]

2.2.3 Local Traffic Study

In South Gloucestershire, traffic and congestion have increased significantly more than the national or county average, especially in the northern fringe of the Bristol area, due to the location of the area along the M4/M5/M32 corridors [49]. The University of the West of England is located along this north fringe, supporting over 2,000 staff and 16,000 students. The following map shows the location of the UWE campus in relation to the motorways and the Avon Ring Road/Coldharbour Lane HOV lanes (shown in green).

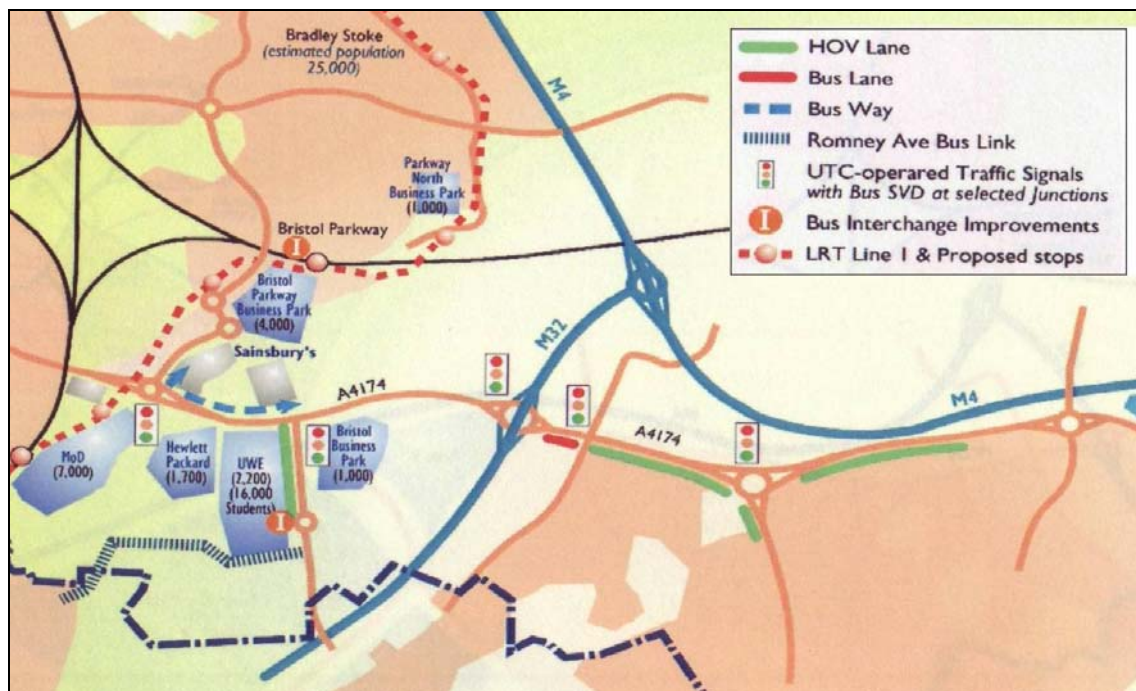


Figure 2.2: UWE Local Traffic Area. Source: [44]

In 1998, HOV lanes were introduced on the Avon Ring Road for use by cars and vans with two people or more, buses, coaches, motorcycles and emergency vehicles. South Gloucestershire Council considered the options, and found that HOV lanes were the best value for money improvement available to decrease congestion. The aim of the project is to give an advantage to car sharers and bus users, i.e. decreased journey times. An online car-sharing database was also setup to allow drivers to find a car-sharing partner.

The Council claims that the HOV lanes have increased traffic flows by over 1000 vehicles (+36%) in the three-hour morning peak period and decreased journey times for all vehicles by 10 minutes. These statistics are surprising because one would suppose that the journey time for single-occupant vehicles would rise, and it is expected that the number of vehicles *decrease* by 10%. Compliance with the new rules has been measured at around 90%, and the AVO (average vehicle occupancy) has increased by approximately 7%. This is slightly lower than the expected increase of 10-15% but reasonable nonetheless. The data received from the Council can be seen in appendix A-1. This data can later be used to calibrate and test the traffic simulation system for use with HOV lanes.

2.3 Traffic Theory

2.3.1 Measuring Traffic Flow

The study of how traffic flows is essential to the design of better road networks. If traffic flow could be completely understood, then traffic levels could be predicted and congestion could be forecasted and avoided. In order to understand this phenomenon, we need to start by looking at what already happens on the roads. Traffic surveys are used to provide measurements of the current situation, and involve counting the number of vehicles going past a point in a certain amount of time. They can be undertaken manually or can be automated using an inductive loop or pressure tube counter. The standard flow/capacity of the average road at optimal conditions is generally accepted to be 2,000 vehicles per hour, per lane. [43]

However, simply knowing the flow rate is not too useful because, for example, 10 vehicles could pass a counter in one minute spaced-out at 60mph or nose-to-tail in a jam. The traffic density describes the number of vehicles in a certain amount of road, and requires the vehicle speed in addition to flow rate for calculation. Density is usually measured in vehicles per km, per lane. The optimal density on a standard road should be around 40 vehicles per km, per lane. At this density, the flow rate should be at maximum, i.e. 2,000 vehicles per hour, per lane. [6]

This method allows one to learn about the traffic flow through a junction or small number of roads, but to gain an overall picture of traffic more is required. Individual trips can be represented as an origin-destination (O-D) matrix, which allows one to represent the number of trips to and from any two destinations. The O-D matrix is an $n * n$ array, where n is the number of origins or destinations considered. [43]. More detailed data can be gathered by examining the daily activities of individuals, however this is outside the scope of the project. [28]

2.3.2 Modelling Traffic Flow

Physicists have been trying to describe the phenomena of traffic for at least half a century. In the 1950s, James Lighthill, an expert on the physics of fluid flow, suggested that the flow of traffic on a

road was akin to the flow of liquid in a pipe. This theory (the Lighthill-Whitham-Richards model) represented the flow of traffic entirely with mathematical equations, and ignored the individual drivers. This sort of model is called macroscopic, and can often produce realistic output, but lacks the complexity to model realistic driver behaviours. [6]

The next approach was to treat vehicles as individual units instead of a continuous flow, and see what behaviour emerges when the vehicles are given simple rules to follow. Each vehicle would move according to the vehicle ahead, speeding up or slowing down to match its speed while maintaining a safe distance between cars. This is a type of microscopic model, which can vary in complexity depending on the aims of the simulation. One well-known model is a cellular automata model designed by Nagel and Schreckenberg. It was very simplistic and followed mainly the rules above, yet exhibits complex phenomena found in real traffic, as described below. [6], [27]

The results from these models and from traffic studies show that flow rate and traffic density are linked in an interesting way. Normally, flow rate increases as density increases, that is, more vehicles are on the road without any having to slow down. However, when the density reaches a so-called ‘critical density’, the flow rate begins to decrease and the traffic becomes congested. An interesting observation is a hysteresis¹ effect that as the density increases above the critical density it is possible for the flow to continue to increase in a metastable or bi-stable state. In this state, any hiccup in the flow can cause the traffic to become congested. [6], [28]

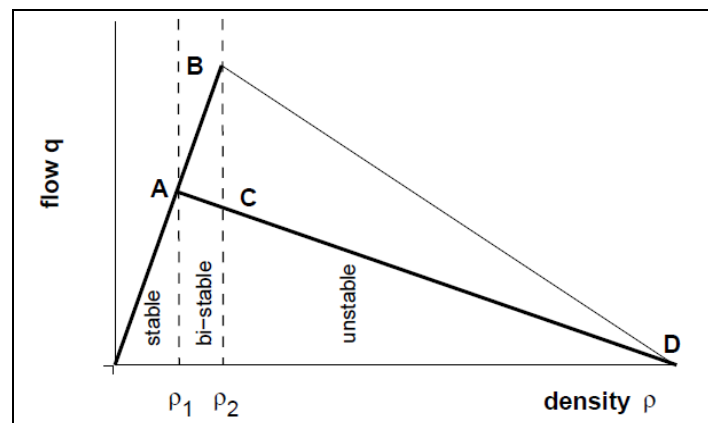


Figure 2.3: Stylised traffic flow-density diagram. Source: [28]

Figure 2.3 shows this fundamental relationship between flow and density. Point A represents the critical density as described above. As the traffic gets denser past this point, either it breaks down into a congested state towards point D, or it continues to flow in a bi-stable state until point B, where some anomaly causes it to become congested. It is worth noting that once the flow has

¹ Hysteresis is a property of systems that exhibit delayed response to state changes, seen as a lag between states.

moved to the unstable state, it cannot return to the bi-stable or free-flow state until the density is lower than the critical density (A). That is, the hysteresis effect happens in one direction only, as the density increases. These conditions are also observed in solids, liquids and gases during phase transitions. This effect can be achieved in cellular models by implementing a ‘slow-to-start’ rule. This is done by making traffic accelerate slower out of stopped traffic than at any other time. [28]

An effect of all this modelling is a better understanding of why traffic jams tend to form with no apparent cause. When the density exceeds the critical density, at any point a fluctuation can cause start-stop traffic. Traffic jams of this form (with no bottleneck cause) move upstream through the traffic, which is easily observed on a graph of space-time showing the position of each vehicle in time (see Figure 2.4). Until the density decreases, the traffic jam will live on, often get worse and sometimes split into multiple regions of stop-start traffic. [6]

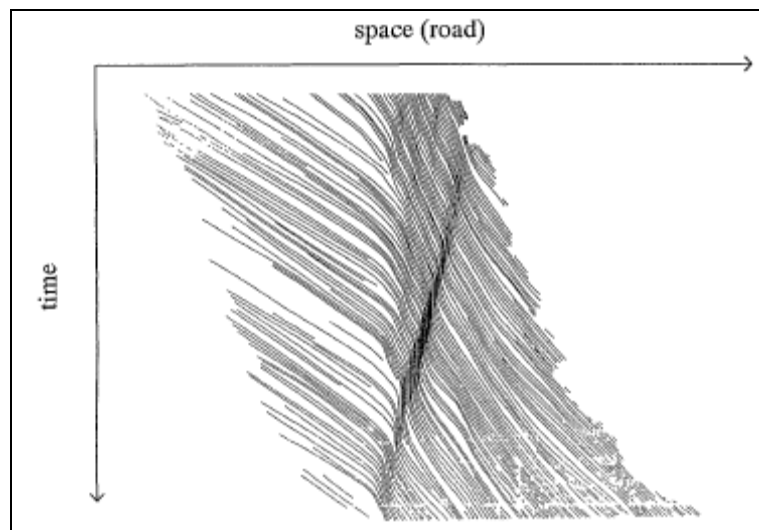


Figure 2.4: Space-time diagram where lines represent vehicles, observed using aerial photography. Source: [27]

Therefore, we can model a traffic jam on a motorway with the free flow, metastable and congested state transition, however real traffic exhibits more behaviours than these states provide. Kerner and Rehborn proposed that there are two different states of congested traffic: one is the traffic jam where no one moves (point D on Figure 2.3) and the other being a ‘synchronised’ state. This state is possible if the speed of all the traffic is more or less the same, and no vehicles stop completely. In this state, high-density traffic can maintain a reasonably high flow rate without breaking down into stop-start traffic (see Figure 2.5). This state can be reached from either the free-flow or metastable states discussed above. Synchronised traffic can be modelled by incorporating a ‘comfort factor’ where drivers aim to drive smoothly with less hard acceleration or deceleration. Researchers still debate the conditions that lead to synchronised traffic; it often occurs at road features such as entry and exit ramps, but sometimes occurs out of nowhere in normal conditions. [6]

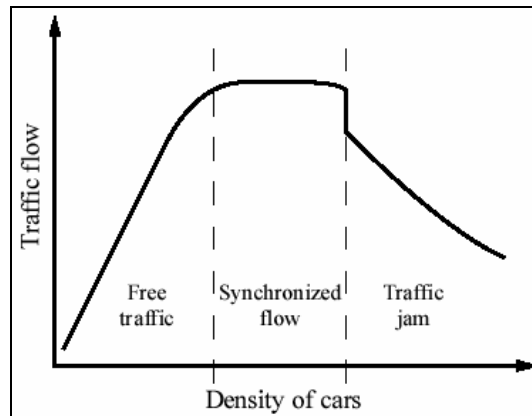


Figure 2.5: Flow-density diagram incorporating synchronised flow. Source: [33]

2.3.3 Understanding Drivers

Drivers are difficult to analyse because they are human and therefore unpredictable and individual. In general, drivers are not aware of the global state of the traffic system, and they only perceive local conditions. Drivers are also emotionally complex, and can change their behaviour for no apparent reason. One problem with many schemes is that they rely on drivers heeding the advice given to them. For example, on the M25 motorway, variable message signs are used to alter the speed limits of the carriageways to optimise traffic flow. Without penalties such as speed cameras, many drivers would simply ignore these limits and decide that they know better. Drivers cannot always be expected to do what seems rational, but for the purposes of this project, it will be assumed that they regard safety with utmost importance, and otherwise try to maximise the utility of their situation through their driving decisions.

Chapter 3: Traffic Simulation

3.1 Introduction to Traffic Simulation

Champion et. al [10] describe simulation as an effective tool used for reproducing and analysing a broad variety of complex problems, difficult to study by other means that might be too expensive or dangerous. Traffic can be viewed as a complex system [Faieta et. al., Sanford in [17]], therefore simulation is a suitable tool to analyse traffic systems. Traffic simulation is the state-of-the-art method used to assess and evaluate transport schemes for reducing congestion [13]. Rather than implementing a scheme without knowing whether the outcome will be a success, the scheme can be implemented in a simulation to determine its effectiveness:

Infrastructure improvements are very costly and each modification must be carefully evaluated for its impact on the traffic flow. Computer traffic simulations form a cost effective method for making those evaluations.

[16]

A study by Leeds University found that the users of traffic simulators include research organisations (including simulator developers), road authorities, consultants and manufacturers. The most frequent applications of traffic simulation are for design and testing of control strategies and evaluation of large schemes. Other applications include on-line traffic management, evaluating product performance, research and education. The majority of these users regard simulation as a necessary or useful tool for the applications listed above. [1]

The APAS assessment of road transport models and system architectures identified four major uses of traffic simulation: simulating networks including interactions between vehicles and new responsive control and information systems, short term forecasting, enhancing assignment models and providing inputs to car driving simulators [5]. The scenarios simulated most often are corridors (areas containing main transport lines) and intersections, followed by city and single-road simulation. Most users utilize simulation to investigate the present situation or the near future (1-5 years ahead) and a few users are interested in longer-term planning. [1]

3.2 Simulation Classifications

Ferber [18] describes simulation as a very active branch of computer science, which consists of analysing the properties of theoretical models of the surrounding world. There are many types of computer simulation; the relevant classifications are summarised below.

Stochastic simulations use random number generators to model chance and randomness. It is unlikely that two runs of a stochastic simulation would be the same, but most generators use seeds that can be set to produce the same set of numbers each time, making reproducible results possible. Deterministic simulations are those that are inherently predictable, always producing the same output for a given input. Deterministic models are useful for experiments where the results need to be reproducible, however most real-world phenomena such as traffic have some degree of chance and therefore require stochastic simulation. [10]

Simulations can be classed as continuous or discrete. Continuous models take the form of equations using variables that correspond to real values. By solving the equations, the state of the model at any given point in the simulation can be calculated. Discrete simulations represent reality by modelling the state of the system and its state changes after time or events have passed. There are two types of discrete simulation: discrete time models and discrete event models. Discrete time models (time-sliced) are those that split the simulation into fixed time intervals. At each interval, the state of the model is updated using functions that describe the interactions. Discrete event models (event-oriented) are those which maintain a queue of events scheduled to happen in order of time, each event representing the change of state of an element in the model. The simulator processes the events in order, and each one can alter the event queue. Section 3.6.4 looks at the implementation issues presented by discrete time and discrete event simulation. [40], [1], [10]

3.3 Macroscopic vs. Microscopic

Traffic simulators can be microscopic or macroscopic depending on the level of detail required. Macroscopic simulators model the flow of traffic using high-level mathematical models often derived from fluid dynamics, thus they are continuous simulations. They treat every vehicle the same, and use input and output variables such as speed, flow and density. These simulators cannot differentiate between individual vehicles, and usually do not cater for different vehicle types. They lack the ability to model complex roadways, detailed traffic control features or different driver behaviours. [16], [8], [31]. Macroscopic simulators are most useful for the simulation of wide-area traffic systems, which do not require detailed modelling, such as motorway networks and interregional road networks [40]. This approach is not very realistic because in real life there are many different types of vehicle driven by different individuals who have their own styles and behaviours. However, it is fast and can be useful and accurate, but is not suited to urban models. [16]

Microscopic simulators model individual entities separately at a high level of detail, and are classed as discrete simulations. Each vehicle is tracked as it interacts with other vehicles and the environment. Interactions are usually governed by car-following and lane-changing logic. Rules and

regulations are defined to control what can and cannot be done in the simulation, for example speed limits, rights of way, vehicle speed and acceleration. [40], [31]. Traffic flow details usually associated with macroscopic simulation are the emergent properties of the microscopic simulation. Microscopic simulators can model traffic flow more realistically than macroscopic simulators, due to the extra detail added in modelling vehicles individually [16]. Microscopic simulators are widely used to evaluate new traffic control and management technologies as well as performing analysis of existing traffic operations [31].

A very simple form of microscopic simulation is cellular simulation, which involves modelling the road as a series of cells and moving the vehicles between cells based on vehicle parameters. This method can implement links using an array with length equal to the number of cells in the link. Cell length has to be determined and must be the same for all cells, which is a disadvantage because it assumes all vehicles occupy the same amount of space. When the simulation is run, each cell can be either empty or occupied by one vehicle. Vehicles are moved forwards by their speed and are restricted by vehicles in front. Links are connected to nodes and rules exist which determine where vehicles go when they reach a node. This method can be very efficient because of the simple array structure, but it lacks some realism. [45], [28]. An even simpler approach is queue-based simulation, where vehicles always move at a set speed until they reach a queue at the end of each link. [27], [26]

3.4 Simulation Scenarios

Traffic simulations can be broadly classified by the type of road network and features they can simulate. The two main classes for simulators are those designed for motorway and urban environments. Simulators supporting a motorway environment focus on multiple-lane high-speed motorways. Much of the complexity required for a city environment does not need to be modelled, and the simulation can focus on vehicle behaviour and interaction. Motorway environments can be simulated accurately by both macroscopic and microscopic simulators. [38]. The main features of a microscopic motorway simulator are car-following and lane-changing behaviours. Junctions are sometimes modelled, allowing entry/exit rate to be varied to test the efficiency of the motorway under varying traffic load. Practical uses include studying the effect of motorway accidents, stop-start congestion, speed limits, ramp metering and lane closures on traffic flow. [9], [38]

An urban environment is one of the most difficult and complex traffic scenarios [16]. In contrast to motorway environments, urban environments have a traffic flow that is interrupted by intersections, traffic lights, roundabouts and other features. In addition to the extra road features, realistic urban simulators should model not only different classes of vehicle, but also pedestrians, cyclists and public transport systems. [14]. Urban traffic networks are usually very complex with many road sections and intersection points, often with conflicting traffic flows [25]. They usually

have to manage a large number of vehicles on small road sections, which can result in a large amount of congestion [48]. Microscopic simulators are well suited to urban environments as vehicles can respond individually to the road features. Macroscopic simulators are not able to model the complexity of urban environments; they are only used to provide abstract flow details.

Some simulators can model both motorway and urban environments at the same time; these are classed as integrated or combined simulators. This is useful for the simulation of large areas encompassing both motorway and urban roads, especially where the performance of one affects the other, and is advantageous to the user as one package can simulate various scenarios. [34], [8]. Some simulators have focussed instead on modelling specific objectives such as to test intelligent vehicle control units, to analyse vehicle safety and comfort, or traffic at toll booths [1]. This project will focus on microscopic urban simulation, as it enables the modelling of congestion in urban scenarios.

3.5 Modelling Issues

Defining the model is one of the first stages of building a traffic simulation. This involves deciding how to represent objects (e.g. vehicles, traffic lights and pedestrians) in the simulation, and what parameters each object will require. It also involves determining how to represent the environment (e.g. road, lanes and intersections), and the effects it has on the other objects.

3.5.1 Modelling Vehicles

In a simulation, vehicles and drivers would most likely be modelled as one entity. However, in the real world they obviously are not, so when deciding on how to model them it makes sense to look at each in turn. Modelling a vehicle is quite straightforward; a few parameters can describe its features and behaviour: maximum speed, maximum vehicle acceleration and deceleration. Acceleration is especially important as it affects the rate of queue discharge. Dimensions are often implemented, enabling trucks and buses to be distinguished from cars. During simulation, current position and heading in the environment are required to keep track of the current state. Other details are not usually required because they add to the complexity without improving realism, however some simulators provide a detailed physics or engine models; these are useful when investigating emissions or the detailed effects of impact. [40], [1]

3.5.2 Modelling Drivers

Paruchuri et. al [32] suggest that the decisions drivers have to make can be split into micro and macro goals. Macro goals are the destination and route taken, while micro goals involve decisions at each point of time in the interest of achieving the macro goal. The macro goal involves daily planning and route generation functionality, often input from O-D data. Micro goals are decisions involving controlling the vehicle, such as desired speed, overtaking and turning. Drivers all have

different driving styles, which are governed by their individual characteristics, such as aggressiveness, confidence and driving experience. Driving styles are often approximated using parameters for important behavioural features and then these values are used as part of the reasoning process to work out the decision a driver would take at any instant. [16], [32]. Examples of these parameters are:

- Preferred speed; the speed the driver likes to maintain on an empty road.
- Preferred rate of acceleration, which is not usually the vehicle's maximum rate, but a rate influenced by a driver's aggressiveness and desire for comfort among other characteristics.
- Preferred rate of deceleration, which is similar to that for acceleration.
- Gap acceptance, which is the distance a driver prefers to leave between them and the vehicle in front.

These parameters can model a fair amount of driver characteristics. An aggressive driver would have a high preferred speed, a high preferred rate of acceleration and deceleration, and a small gap acceptance. A cautious driver would be much the opposite. Many simulators incorporate all of these parameters, and demonstrate realistic driving behaviour. To convert these parameters into actions, a reasoning process must be evaluated at each stage of the simulation; this is discussed in section 3.6 below. [16], [32]

3.5.3 Modelling the Environment

In traffic simulation, the environment in which vehicles drive is a road network which is made up of link segments, nodes (junctions and intersections) and control features which are usually part of the node. Each link can have one or more lanes, and may operate in one or both directions. Nodes can be broken down into the following basic types: uncontrolled non-priority junctions, priority junctions, roundabouts, signal controlled junctions and grade separation junctions. Special cases of link exist such as merges where two lanes must merge into one, diverges where one lane splits into two and weaving sections where a diverge closely follows a merge. Examples of other control features are traffic calming schemes and pedestrian crossings. [43]

Models usually represent the network as a collection of links joined by nodes. Nodes have properties such as location (e.g. coordinates) and type of node. Links have properties such as length, number of lanes, speed limit, etc. For road networks that are not self-contained, sources and sinks are created as positions in the network where vehicles arrive and leave respectively. Sources have parameters that determine the rate of entry of vehicles into the network, and sinks always remove vehicles that reach them. [28]

3.6 Simulation Issues

Models of vehicles, drivers and the environment are of no use unless they can be manipulated during simulation. Simulation involves using behavioural rules to change the model over time. This involves moving each vehicle based on its parameters and its driver's decisions. To simulate a single car on a long straight road is rather simple, as the vehicle would just accelerate to the driver's preferred speed. It becomes increasingly complex as other vehicles, more lanes, and more roads are introduced; to deal with these, vehicle following and lane changing models are used. [24]

The heart and soul of a microscopic traffic simulation is the car-following and lane-changing logic.

[31]

3.6.1 Vehicle Following

Vehicle following models are those that attempt to describe how a vehicle behaves in standard conditions. These models are called such because they focus primarily on the relationship between the current car and its leader [40]. Many vehicle following models are discussed in the literature, and many of these are based on anti-collision concepts. One well-known model of this type is the Gipps model, developed in 1981. It assumes that a vehicle always aims to be able to stop safely if its leader performs an emergency stop. It calculates the new speed of the following vehicle using the current speed of both vehicles, the following driver's desired speed, maximum acceleration and deceleration, and the gap between vehicles. It calculates two possible values for speed; one based on the driver's desired speed limited by vehicle performance, and one based on safety, i.e. the speed that must be taken to ensure there is no collision. It then selects the minimum of these possible speeds to use in the simulation, which ensures both limitations are applied. [11]

Another class of models are known as psycho-physical vehicle following models, proposed by Wiedemann in 1974. These models attempt to capture both physical and human components of vehicle control. They do this by maintaining a vehicle state, where the current state is determined through differences in speed and distance to the leading vehicle. Each state has a different method of calculating the acceleration of the vehicle, and upon a state change, the new acceleration is calculated once and is then constant until the next change of state. Since the acceleration is not calculated at each time step in the simulation, it has been shown that the event-oriented time advance method (see section 3.2) is very efficient for this model. State changes can be determined and listed in a queue so they can be processed in order. Some examples of vehicle states and the situations that cause them are shown in the observation-decision diagram below. [40]

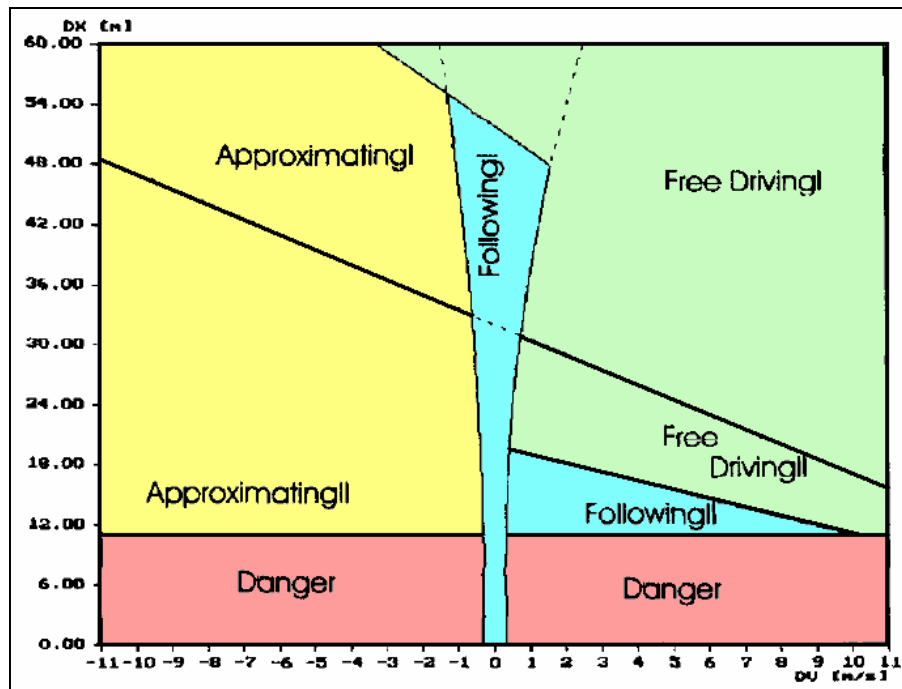


Figure 3.1: Observation-decision diagram showing limits in different of speed (DV) and distance (DX) for selecting vehicle states. Source: [40]

Fuzzy logic can be used to model driver inaccuracy and uncertainty, and can lead to more realistic models than those purely based on accurate mathematics [4]. Vehicle following models range from simple relationships between speed and distance to complicated models of driver behaviour and emotion. All of these can model the main features of the macroscopic traffic flow with acceptable accuracy. [24]

3.6.2 Lane Changing

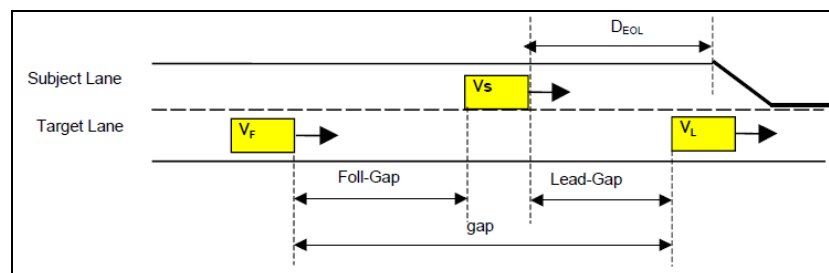


Figure 3.2: Basic notation for lane change manoeuvres. Source: [20]

Lane changing models are those that describe when and how vehicles change lanes on a multi-lane road such as a motorway. A lane change scenario can be summarised as in Figure 3.2. The behaviour of each vehicle depends on the vehicles around it, and each vehicle has to make decisions based on its knowledge of the environment [20]. Lane changing models have been studied in the literature to a similar extent as vehicle following models, and several important models have been devised [24]. The basic lane changing rules are described in the following quote.

All lane changing rules, no matter if for CA or other models, follow a similar scheme (e.g. Sparmann, 1978): In order to change lanes, drivers need an incentive, and the lane change needs to be safe. An incentive can be that the other lane is faster, or that the driver eventually needs to make a turn. Safety implies that one needs enough space on the target lane.

[28]

Using this model produces a lane-changing system made up of two rules: incentive and safety. If both are true, then the lane change takes place. Incentive could be true if the gap in front of the vehicle in the other lane is larger than the current gap, in which case an immediate speed increase would be possible. Safety can be determined by checking whether the gap behind on the target lane is large enough that the vehicle behind doesn't have to brake immediately. This has been shown to model U.S. traffic quite well, but there are other issues to consider, e.g. vehicles may need to change lanes in order to be in the correct lane at the intersection or motorway exit. [11]

Other lane change issues emerge from road rules, for example in the U.K. passing is normally only allowed on the right and vehicles should try to keep to the left lane if possible. Driver characteristics can also play a part, since some drivers prefer to stay in the same lane even if changing lanes would result in a higher speed, and some drivers prefer to stay in outer lanes even if there is space in inner lanes. [11], [28]. Details of some more complex recent lane changing models can be found in [20] and [24].

3.6.3 Urban Simulation Issues

Specific behavioural rules are required to achieve realistic traffic simulation in an urban environment. Some examples of features that can be implemented are traffic lights, roundabouts, traffic signs, obstacle avoidance, overtaking using the oncoming lane, turning into side-roads and turning at junctions. These can be implemented using specific vehicle behaviour rules for each feature, which need to interoperate well with the vehicle following and lane changing behaviours. The rule that takes precedence can be decided by the situation and priority weighting of each rule, so that time- or safety-critical incidents are dealt with immediately. [16], [38]

3.6.4 Accuracy and Time Issues

The two types of discrete simulation mentioned in section 3.2 are discrete time models and discrete event models. The accuracy and run time of microscopic simulators depend to an extent on the method of time advance implemented. The accuracy of discrete event simulators is perfect, because the exact simulation time of each event is calculated to determine its position in the event queue. However, the run time of such a simulation depends on the number of events that need to be processed. This in turn depends on the number of dynamic objects (e.g. vehicles) in the simulation.

It is not usually possible to predict the number of events which will occur and therefore not possible to predict the run time of such a simulation. [40]

The accuracy and run time of discrete time simulators depends on the number of dynamic objects and the chosen time interval. The value used in many existing simulators is 1.0 seconds, which corresponds to the average reaction time of the driver. This means that every one second in simulated time, the behaviour rules are evaluated and all vehicles positions are updated. The smaller the time slice, the more times every object must be evaluated and therefore the longer the overall run time. Smaller time slices give results that appear more accurate because the vehicle interactions are evaluated more often. This means there is a trade-off between run time and accuracy when using discrete time simulation. [40]

A study by Schulze and Fliess [40] evaluates discrete event and discrete time methods in traffic simulation. It compares event based to time sliced with values of 1.0, 0.5 and 0.2 seconds. It recommends event based methods for use in psycho-physical vehicle following models and concurs that this is both faster and more accurate than time sliced methods. It shows that a smaller time slice gives more accurate (closer to the event-oriented result) and more consistent results than a large time slice, and suggests that a time slice value of 0.2 seconds is preferable especially when simulating high volumes of traffic. It is possible though, that while the event-oriented method is accurate, it is not realistic, as it does not model driver reaction time or fuzzy driver perception.

3.7 Requirements of Traffic Simulators

The SMARTTEST project conducted by Leeds University performed an in-depth study on the importance of various features for traffic simulations. The study identified 58 microscopic simulators from across the world, including commercial, free and under-development systems. Of these, 32 were chosen for in-depth analysis, shown below as classified by their developers. [1]

<i>Motorway</i>	<i>Urban</i>	<i>Integrated</i>	<i>Other</i>
CASIMIR	AUTOBAHN	AIMSUN2	ANATOLL
DRACULA	FREEVU	CORSIM	PHAROS
HUTSIM	FRESIM	FLEXSYT II	SHIVA
MICSTRAN	MIXIC	INTEGRATION	SIMDAC
NEMIS	SISTM	MELROSE	
NETSIM		MICROSIM	
PADSIM		MITSIM	
SIGSIM		PARAMICS	
SIMNET		PLANSIM-T	
SITRA-B+		TRANSIMS	
SITRAS		VISSIM	
THOREAU			

Table 3.1: Selection of Traffic Simulators by Type. Source: [1]

Most of the simulators use a time-sliced approach as described in section 3.6.4. The developers of the above simulators completed a questionnaire for SMARTEST to discover the features they have implemented. A similar questionnaire was completed by 44 known users of traffic simulation tools to identify the most important requirements. The following chart shows the common features identified by SMARTEST along with the percentage of simulators that implement each feature and the percentage of users who thought it was crucial or important. It is in decreasing order of importance as rated by the users of simulation systems, apart from queue spill back and weaving which were not on the users' questionnaire but are implemented by most systems in any case. [2]

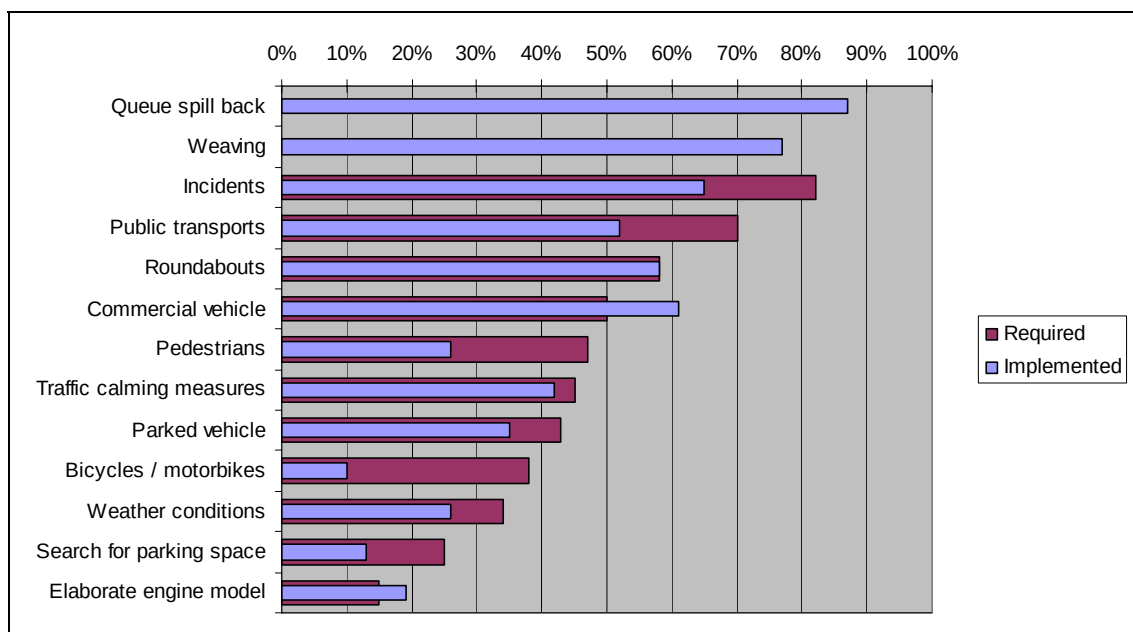


Figure 3.3: Importance of Traffic Simulator Features. Data sources: [1], [2]

Note that the ability to model road networks, vehicle following, lane changing and intersections are essential features and therefore are not included. Although the two data sets are not intended for comparison, it gives an idea of the percentage of developers versus users who consider each feature important. The data shows that the most required features are incidents, public transports and roundabouts. A few features appear to be more important to the users than to some developers: incidents, public transport, pedestrians and bicycles/motorbikes; perhaps these features deserve more attention in future simulators. Only one of the respondents (PARAMICS) mentioned the ability to model HOV lanes, although this was not explicitly asked by the questionnaire.

The study found that a user-friendly graphical interface is crucial for both input and editing the model and presentation of results. The majority of uses are to simulate traffic over a 0.5 to 2 hour period, i.e. the peak rush hours. The required speed of simulation execution is faster than real-time; about half the users are content with 1-5 times faster while the remainder require over 5 times faster. The most important model properties identified were that the model must have been verified, and that it allows user-defined key parameters and provides defaults for these parameters. Simulators provide results in the form of macroscopic indicators; the required and implemented indicators are shown below in the same form as the previous chart. [2]

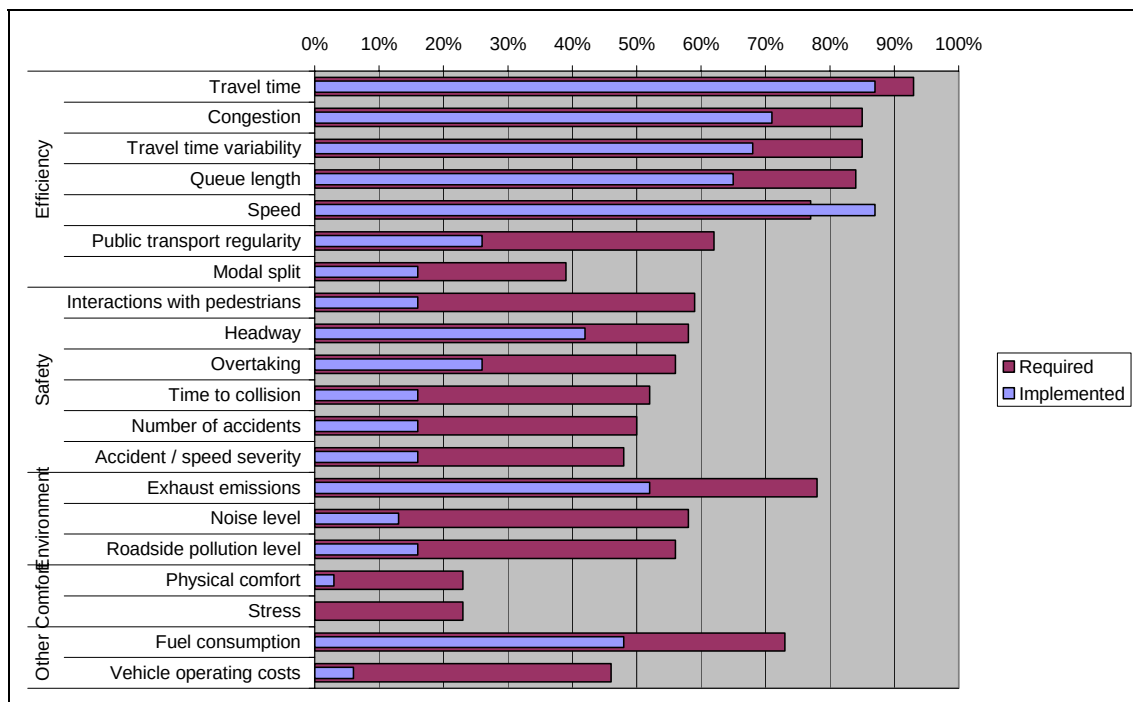


Figure 3.4: Importance of Simulator Indicators. Data sources: [1], [2]

It is clear that many indicators are considered less important to developers than to users; many in the area of safety, environmental and public transport. The most required efficiency indicators are provided by most simulators but are still omitted from many. These could be areas of improvement

for indicators of future traffic simulators. Features in the transport telematics¹ and technological functions categories and are shown in the chart below, and also leave room for improvement. The most important functions are concerned with urban traffic control, such as traffic signals. Public transport support is still lacking in simulators although it is considered of great importance to users.

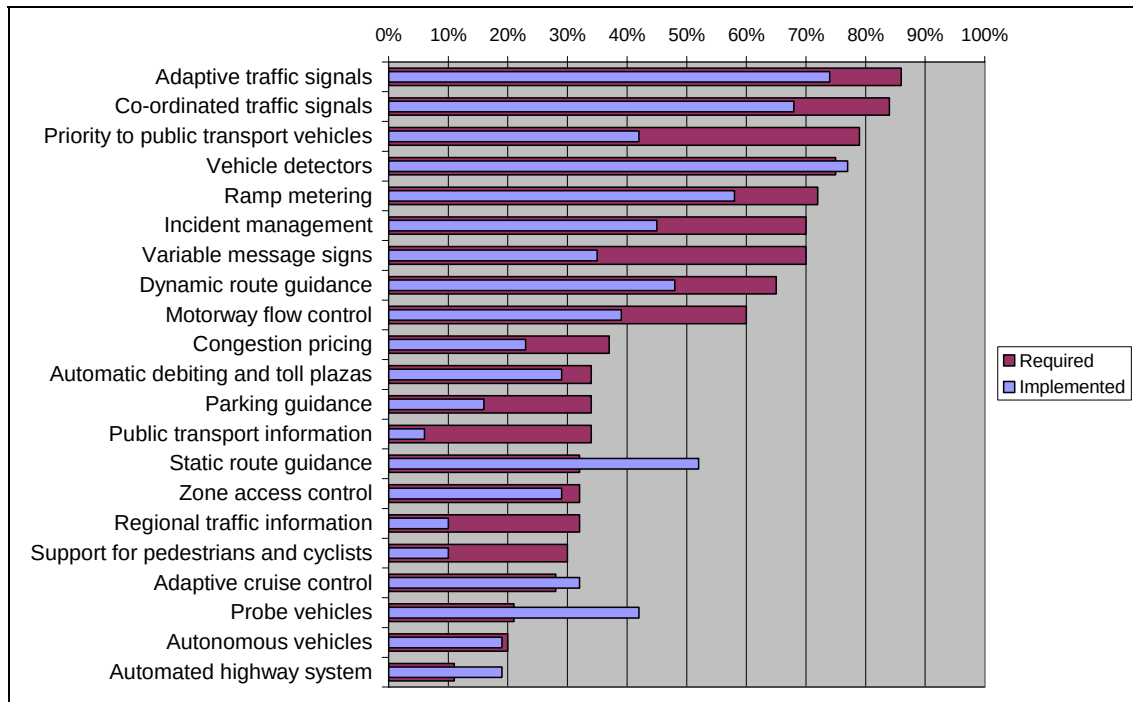


Figure 3.5: Transport Telematics and Technological Functions. Data source: [1]

¹ Transport telematics is the combination of telecommunication and informatics, i.e. communication and information systems for optimising transport.

Chapter 4: Agent-Based Traffic Simulation

4.1 Multi-Agent Systems

4.1.1 Agents

Agent-based systems are a relatively new concept in computing, having been studied since about 1980, and are currently of great interest to researchers. Agent systems are all about autonomy; building systems with some degree of intelligence. Although there is no universally accepted definition, one definition of the term ‘agent’ is presented below.

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.

[56]

Agents are computing entities that receive sensor input from their environment, and can produce actions that influence their environment. Agents usually have a set of actions they can perform, each action having preconditions and aiming to affect their environment. Agents can communicate with each other, allowing them to cooperate, negotiate and coordinate their actions. Agents have objectives and goals, can make plans of actions to perform, and aim to achieve their goals through these plans. [18]

A multi-agent system (MAS) is one in which a number of agents exist and interact in the same environment. Each agent is able to perceive and influence part of the environment; this area is known as the agent’s sphere of influence. The environment can contain passive objects that can only be acted on by agents and therefore have no control over themselves. [18], [56]. In agent-based traffic simulation, the environment is the road network, and the agents are principally vehicles who can affect the agents surrounding them. Control features such as traffic lights could also be modelled using agents.

4.1.2 Agent Environments

Russell and Norvig suggest that agent environments can be classified by their various properties, which are summarised below. [56]

- An *accessible* environment is one in which an agent can reliably obtain information about the current state of the environment. Environments can be inaccessible if there is an amount of uncertainty or if some information is simply unavailable to an agent.
- A *deterministic* environment is one in which an agent’s actions has guaranteed effects. The agent can be certain of the effect of its actions without worrying about failure. Most non-

trivial agent environments are non-deterministic; therefore, agents must be able to cope with failed actions and unexpected circumstances.

- A *static* environment is one that will be unchanged except by the actions of the agent. A *dynamic* environment is one that can be altered by other processes such as other agents. Most environments are dynamic, therefore agents must be reactive, i.e. they must be able to respond to changes in the environment.
- A *discrete* environment is one that has a finite number of states, whereas a *continuous* environment has an infinitely large number of states.

The most complicated type of agent environment is one that is inaccessible, non-deterministic, dynamic and continuous; these can be referred to as *open*. Most agent environments are also real-time, in the sense that time plays a part in an agent's interactions in that environment. In a real-time environment, agents cannot spend an indefinite amount of time deciding the best action and usually aim to make decisions in a short amount of time. [56]. The traffic environment can be described as both open and real-time.

4.1.3 Differences to Object-Oriented Programming

To aid understanding, the agent-oriented approach could be thought of as an extension to the object-oriented (OO) approach. Objects are computing entities that encapsulate some state, have methods to modify their state, and communicate by passing messages. Objects therefore have some form of control over their state, but they do not have control over their behaviour. Agents are considered to have control over their behaviour, meaning that each agent has its own thread of control and can perform actions whenever it chooses without being acted on by any other entity. Although it is possible for all objects to have their own thread of control, it is not central to the OO concept.

Agents also exhibit flexible behaviour, which is not part of the OO paradigm. This means they are capable of reactive, proactive and social behaviour. Reactive means they can perceive and respond to changes in their environment, proactive means they can take the initiative to achieve their goals, and social means they can interact with other agents to satisfy their objectives. The natural evolution of agent-based programming is depicted in Figure 4.1. [56]

	Monolithic Programming	Modular Programming	Object-Oriented Programming	Agent Programming
Unit Behavior	Nonmodular	Modular	Modular	Modular
Unit State	External	External	Internal	Internal
Unit Invocation	External	External (CALled)	External (message)	Internal (rules, goals)

Figure 4.1: Evolution of Agent Programming. Source: [30]

This suggests that agent programming is more declarative than imperative programming. In imperative programming, an explicit algorithm is specified that describes the steps required to reach the goal. Declarative programming, in contrast, specifies the goals and rules that can be used, and leaves the algorithm unspecified. Agents can use intelligence to define their own steps required to reach the goal, based on the rules they have available.

4.2 Agent-Based Modelling (ABM)

4.2.1 Multi-Agent Modelling and Simulation

Simulation can be implemented using multi-agent systems; this is known as multi-agent simulation or agent-based modelling. The agent paradigm maps neatly onto many modelling scenarios because each individual in the scenario can be directly represented as an agent. Behaviours can be programmed into the agents so that individuals behave in the same way as the entity they are modelling. An interesting point is that individuals are given simple behaviours, and when many agents are simulated as a group, behaviours often emerge that were not explicitly programmed into the agents; these are known as emergent phenomenon. [18]

In contrast to the classic approaches, multi-agent simulation does not boil down to implementing a model and then analysing the response from this model as against the input parameters, but participates in the process of searching for models.

[18]

This suggests that multi-agent simulation can be an excellent prototyping tool for developing and refining models. Because it can allow parameters of the individuals and the environment to be varied easily, it is possible to experiment with many alternatives and gain results in real-time.

4.2.2 Calibration and Validation

Calibration is the process of setting model parameters and behaviour so that it matches real-world data. This usually requires data on the microscopic or individual level, such as vehicle speeds,

reaction times and driver behavioural parameters. Calibration also involves optimising the model for the individual scenario to ensure results are realistic. Validation is the process of comparing simulation outputs with real life data to ensure the results are plausible. The data used in validation should differ from the data used for calibration, or inaccuracies may not show up. Validation data is usually macroscopic statistics such as flow rate, speeds and queue time, which can easily be compared with data from real traffic experiments. [3], [28]

A related issue in agent-based modelling is understanding causality. This problem is that unexpected results can be either useful or the result of mistakes or inaccuracies in the model. It can be difficult to understand the cause of certain behaviours, and tools to assist in this are generally lacking. [35]

4.2.3 System Development

The design and development process for multi-agent simulations is inherently different to traditional systems. The focus is on accurate modelling of individual agents rather than the design of passive functions. When designing an agent, the important decisions involve its behaviour, actions, plans, goals, and how it interacts with other agents and the environment. At the Center for Agent-Based Modelling, it is suggested that modelling is a 7-stage process: brainstorming, theory, hypothesis, flowchart, code, analyse and publish. [46]

Because multi-agent systems are relatively new, there is not a generally accepted methodology for the design and development process, and existing OO methodologies do not cover them to a sufficient extent [12]. However, many new methodologies are being proposed, such as Agent UML, where features of the popular UML standards are being adapted to fit in with the design of multi-agent systems [42]. Other methodologies in use include MESSAGE, PASSI, Tropos, Prometheus and MaSE [30]. For the purposes of this project however, existing development methodologies and UML will suffice, making note of suggestions from the 7-stage process.

Multi-agent systems are often written in an object-oriented language like Java or C++, because agents can be easily viewed as an extension to objects [28]. When developing a multi-agent simulation, it makes sense to use a helper package designed for this purpose. This is because the important part of such a system is the accuracy of the model parameters and behaviour, and a helper package can abstract the design of the model from the programming of it. This allows the modeller to focus on tweaking the model rather than solving unrelated coding issues.

To assist with agent-based modelling and simulation development, there are integrated development environments (IDEs) and toolkits/libraries. IDEs are all-in-one modelling and simulation environments, which allow the model development, any code required, and running of the simulation to all be done in the same graphical environment. Because the IDE can be designed

with agent-based modelling in mind, graphical tools and abstract language can make it easier and more accessible to modellers. Toolkits or libraries intend to make simulation facilities available to existing programming languages such as Java or C++. They are generally less accessible to modellers because the user must be able to program in the toolkit's language to start with, and then must learn how to interface with the toolkit. [47]

Some examples of popular toolkits/libraries are Swarm, Repast, Cybele, JAS and MASON. These are all free open source packages available for Java. They provide efficient simulation facilities and can support any behaviour that can be programmed in the base language. Swarm, Repast and MASON are compared in [35], which concluded that they are fast and powerful but less intuitive and more difficult to learn than NetLogo, an IDE that is described below. [35]. Since the aim of this project is to experience agent-based modelling, the decision has been made to use an IDE so that modelling issues can be paramount rather than programming issues.

4.2.4 Integrated Development Environments

These systems provide a complete environment for developing a model and running the simulation, and offer various levels of abstraction. A screenshot of each environment is shown, along with a brief overview of the system. Only software that is free to use and provides a visual representation of the simulation has been considered.

NetLogo [47]

NetLogo is an agent-based modelling environment designed for simulating complex natural and social phenomena. It is written in Java and is therefore cross-platform, it is freeware, and has a large friendly user group. It enables the user to model any number of agents in a variable-size environment using a simple programming language derived from Logo. It is designed for use by students and researchers to explore the behaviour of programmed agents under varying conditions. It follows the philosophy of 'low threshold, no ceiling', meaning that new users should find it easy to get started, but advanced users should not find it limiting.

NetLogo comes with extensive documentation, tutorials, and over 300 sample models that demonstrate all aspects of the tool. Models can be saved as Java Applets and run on web pages by any user with the Java Virtual Machine installed. It is possible to view the current state of the environment and agents in 2D or 3D, and agents can be given any vector shape to display their type. Commands can be in the form of procedures that are called by buttons on the interface, or entered directly in the command console on the main panel. Its logo-esque language is quite natural, allowing code to be easy to read, write and understand.

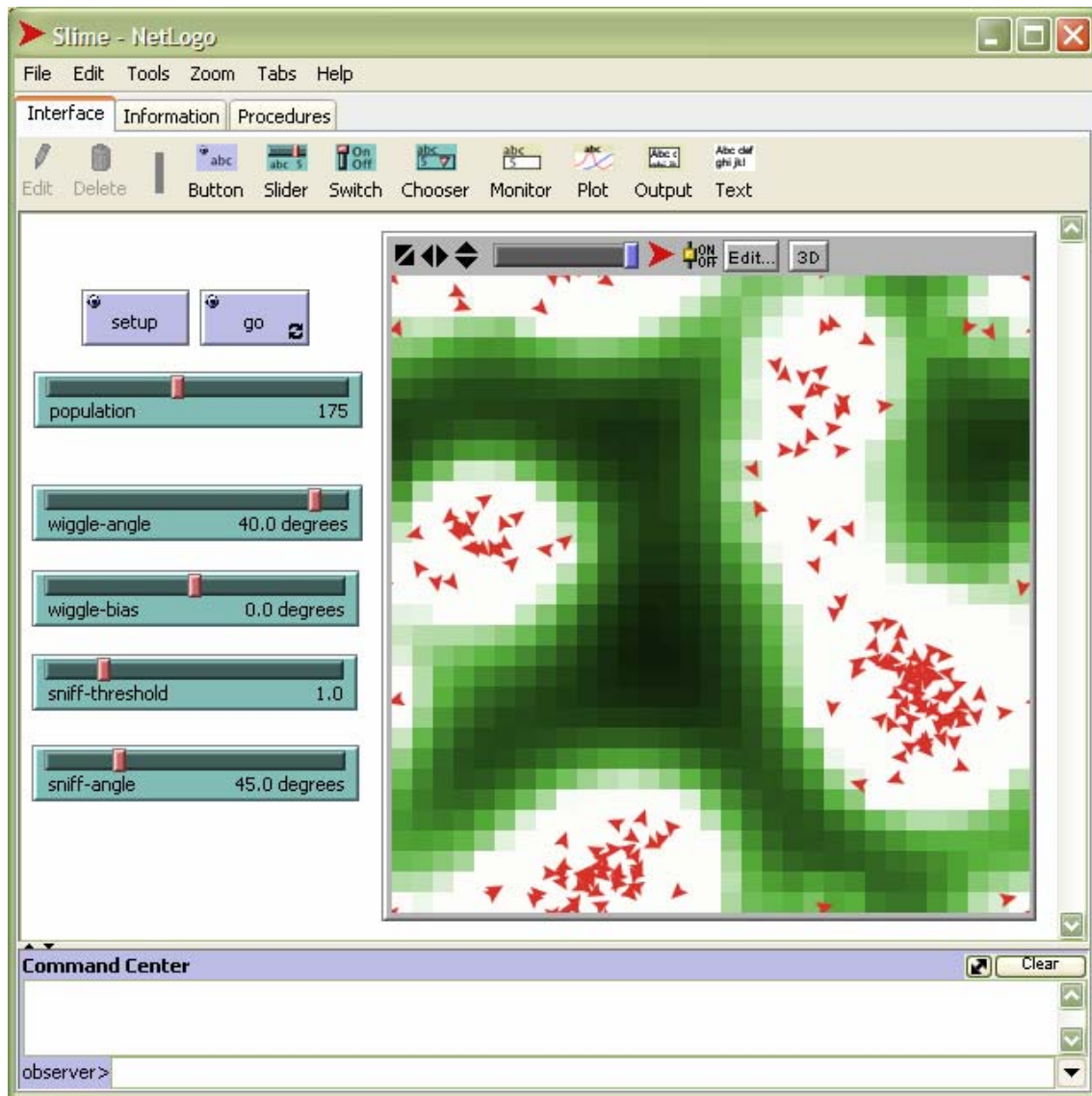


Figure 4.2: NetLogo Interface

NetLogo has a well-designed graphical interface and interface builder in one that allows the novice and expert alike to run, alter and develop models with ease. It provides many built-in widgets to alter simulation parameters at runtime, including sliders, buttons, and drop-down menus, and allows output in the form of graphs and variable monitors. Simulation time is measured in discrete ‘ticks’, and simulation speed can be adjusted by a slider above the display. It can provide deterministic simulation if the random seed is set before the simulation is run.

Breve [23]

Breve is described by the author as a 3D simulation environment for the simulation of decentralized systems and artificial life. It provides simulation of continuous time and space, a physics engine, collision detection, and allows realistic 3D physical simulations to be easily defined by the behaviour of agents. Simulations are programmed in an interpreted object-oriented language called ‘steve’, which is specifically designed for implementation of 3D simulations, and uses natural

language structures to simplify coding. Breve is free, cross-platform, and comes with around 40 example models, which help the user to learn the system.

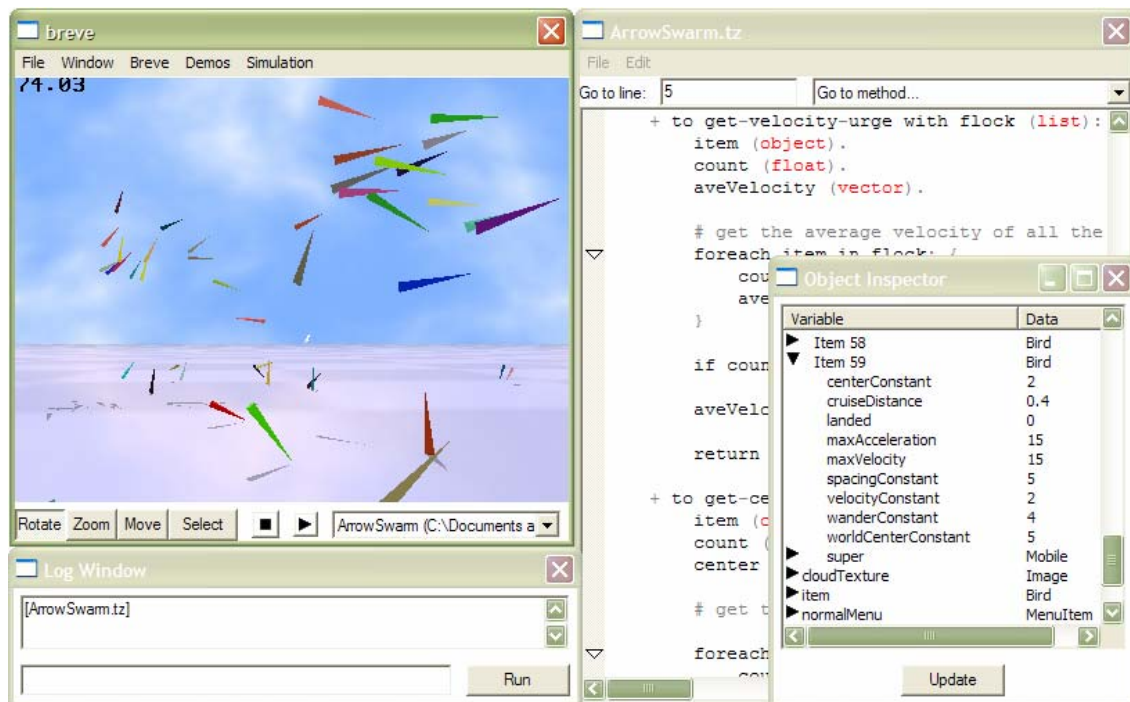


Figure 4.3: Breve Interface

The breve GUI is presented in the form of a window for each part of the simulation. One shows the 3D display with simulation controls and system menu options, and others open for command log, simulation code, and object inspector. The visualised world can be manipulated using the mouse, and objects in it can be interrogated through context menus. This sophisticated system is best for simulating physical entities in an extensive 3D continuous world.

SB-MASE [21]

Subsumption-Based Multi Agent Simulation Environment (SB-MASE) is a cross-platform Java application designed to assist in the development of control software for simulated and real agents, such as robots. It is based on the subsumption architecture, which is an AI concept to describe behaviours. It allows complex behaviour to be split into layers of more and more abstract behaviour, where each layer can use (subsume) the layers below it. SB-MASE allows agent tasks to be modelled as processes, with sensors as inputs and actuators as output.

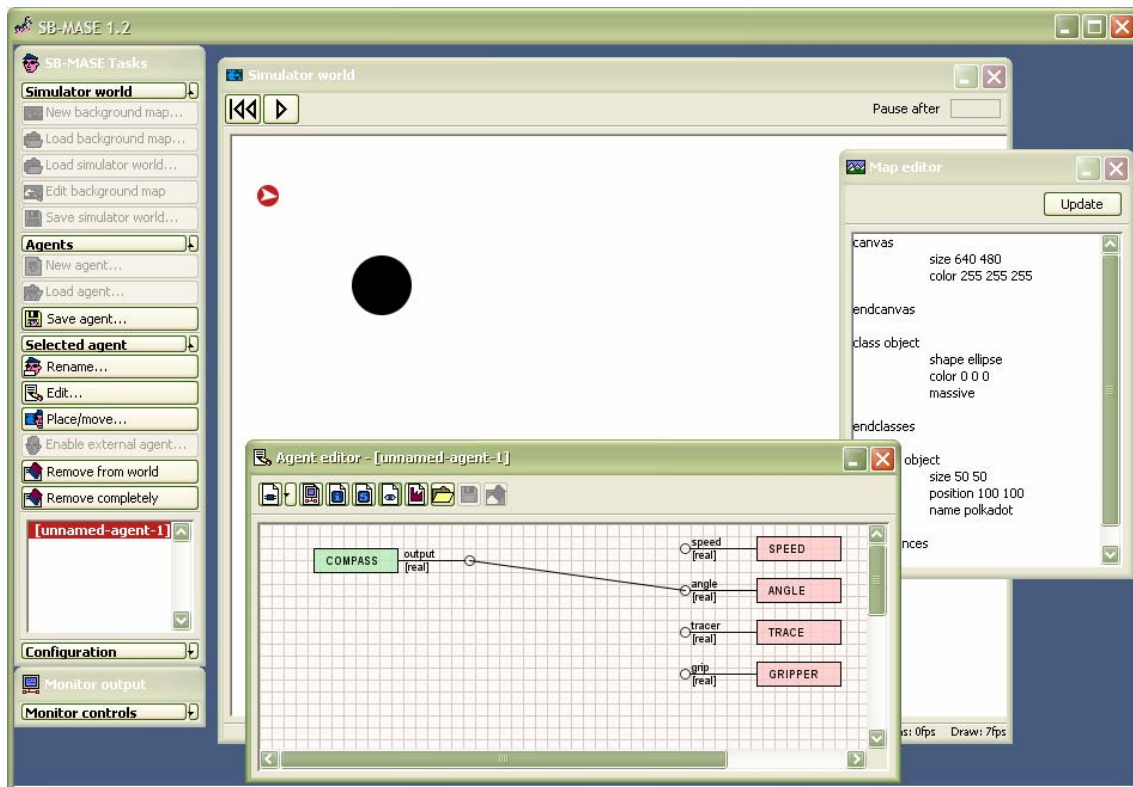


Figure 4.4: SB-MASE Interface

SB-MASE provides a number of windows within a main frame; program controls, simulated world, map editor, agent editor, and more. Although multiple agents can exist in the world, since it is designed to interact with real-world agents, usually only a preset number of predefined agents are used. Agents are inserted and removed explicitly by the user, and their behaviour is designed using a drag-and-drop process editor that can connect outputs from one process to the input of another. This system would be best used to develop control systems for physical agents such as robots.

SeSAm [41]

SeSAm (Shell for Simulated Agent Systems) is an integrated environment written in Java, intended for general agent-based modelling and simulation. It is designed to allow complex models to be easily designed and run in a single modelling environment. Agent reasoning is represented using a UML-style activity diagram, where each activity exists as a script of actions that run predefined functions, and the current state is determined by rules. All functionality can be designed through the graphical interface by selecting operators and methods. The tool currently comes with seven example models to try out.

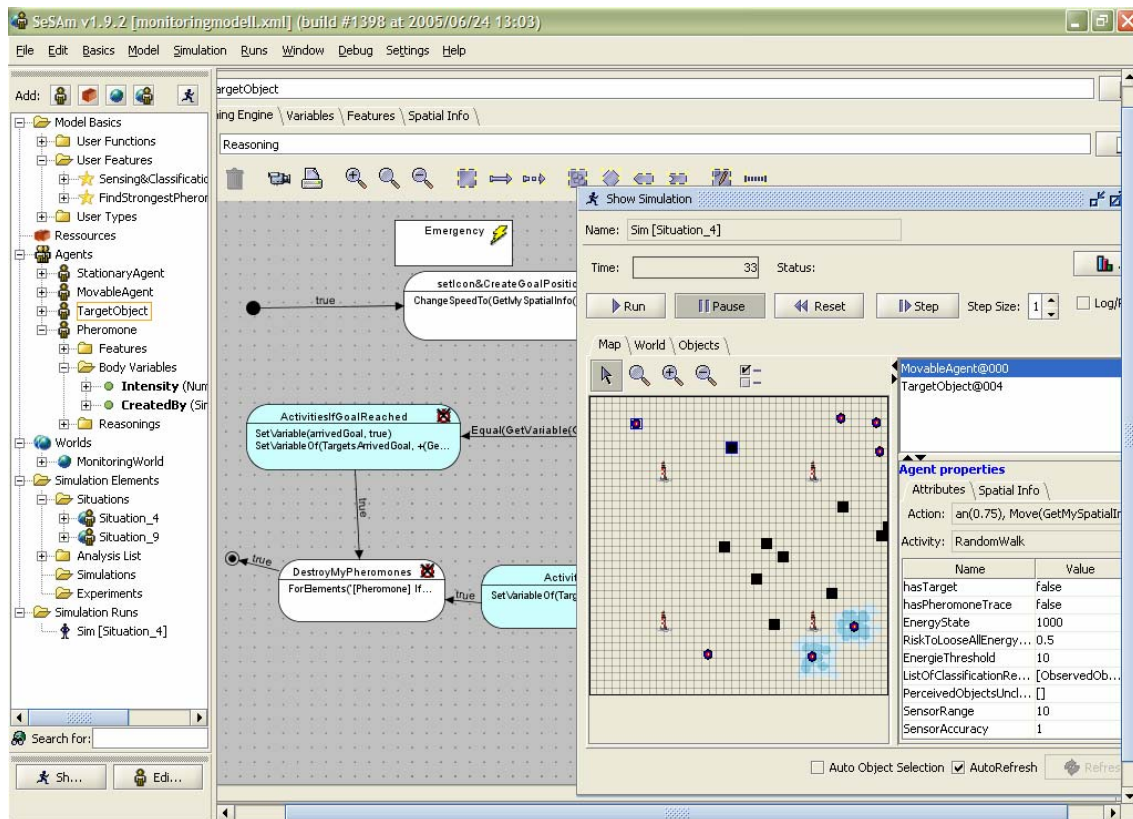


Figure 4.5: SeSAM Interface

Again, the SeSAM interface provides a number of windows within a large frame. All the parts that make up the model are listed on the left, including agents, the environment, functions, variables and simulation scenarios. Double clicking on any of these opens an edit window to modify the item. The agent reasoning interface makes it easy to view the possible states and actions of an agent, although editing the actions seems a little complex.

4.3 Multi-Agent Traffic Simulations

A number of urban traffic simulations have been developed using agent-based modelling, most utilising toolkits/libraries. Many have been developed by researchers and students, as commercial traffic simulators do not yet use the agent approach. A worthy selection of recent agent-based simulators is: SCANer II by Champion et. al [10], Tang and Wan [45], Ehlert and Rothkrantz [16] and Rigolli and Brady [38].

Chapter 5: Design

5.1 Project Design

5.1.1 Project Overview

This project aims to investigate the use of agent-based simulation on the field of traffic engineering with a view towards tackling congestion. Traffic simulation will not *solve* the problem of congestion, however it can help by allowing one to assess, in advance, whether a scheme is likely to work, and how well it compares to other options. The remainder of this report follows the design and development of such a simulator. The project design phase looks at project-wide decision such as the process followed and requirements. The software design phase looks at software-level decisions such as system requirements and UML diagrams.

An agent-based traffic simulation will be designed and implemented making use of a multi-agent simulation package as described in section 4.2.3. The project will be to develop an urban microscopic traffic simulation. It will be tested with both motorway and urban scenarios, and provide an implementation of many features detailed in Chapter 3, such as vehicle-following, lane-changing, network modelling and intersections. Due to the short time frame of this project, features that would be required for a real-world simulator (see section 3.7) will not all be implemented. The most crucial and interesting subset of features will therefore be chosen for development and implementation.

5.1.2 Software Development Process Model

The software development process chosen for this project is rapid iterative prototyping. This methodology involves making a simple functional prototype, then iteratively evaluating and evolving it until it fulfils all requirements. It allows functionality to be added in a systematic manner to a working version of the system, and allows requirements to be assessed at every stage of the development process. It is suited to a system where the development must be fast and the system needs to be evaluated by the user at each stage of development, and to systems where the requirements may not be accurate.

This methodology has been chosen because for the purposes of this project, the author is both the developer and the user, and may therefore have trouble extracting a perfect set of requirements up front. The system must also be developed in a short timeframe, which makes rapid prototyping a good choice of process. After initial requirements analysis, a first prototype will be developed which will provide the functional core of the system. It will only fulfil the most basic of requirements, with a simple interface and no elaborate extras, as a proof-of-concept prototype. This prototype

will be tested and evaluated, and then the iterative process of prototyping begins. After a number of iterations, the final prototype should fulfil all the requirements.

For the purposes of testing, a methodology similar to the V model will be used. The V model matches different levels of test to different design and development phases. This can be applied to agent-based modelling by looking at the various outputs and behaviours shown. An agent-based simulation can be tested at three levels: runtime errors, realistic behaviour of individuals, and valid overall behaviour. Runtime errors can be tested on individual functions by running all types of valid and invalid input into the model. Individual behaviour can be tested by analysing individual agents (on their own and around others) and checking that their microscopic behaviour is expected in all situations. Overall behaviour involves checking that the macroscopic emergent behaviour of all the individuals is valid for all scenarios. Test plans at each level are detailed in section 5.3.

5.1.3 User Analysis & Use Cases

Before use cases can be identified, the users and ultimately actors are identified. From section 3.1, we know that the potential users of a traffic simulator include research organisations, road authorities, consultants and manufacturers. The road authority and consultant users are likely to be traffic engineers who are used to working with models of the road network. They will probably use a CAD package for manipulating the models; ideally, this model should be read directly into the simulator. The simulator's role in the traffic engineer's work would be to input an existing road model and output the results of the simulation asked of it. In this case, the engineer would both define the parameters and run the simulation.

It is conceivable though, that the finished model be distributed and run by other engineers, or by the decision-makers who have to be convinced by the results. Therefore, the actors for this system will be the model developer and the model user. The developer will input the road network, set parameters, and calibrate/validate the model. The user will require a subset of this functionality, and only need to load and run the saved model. It can be assumed that the developer will be familiar with technical terms relating to traffic engineering, whereas the user may not. The users from research organisations and manufacturers are likely to need access to developer-level use cases at all times. They may however be designing test models from scratch rather than importing an existing road network, so this must be supported. A use case diagram and descriptions of business-level tasks follows.

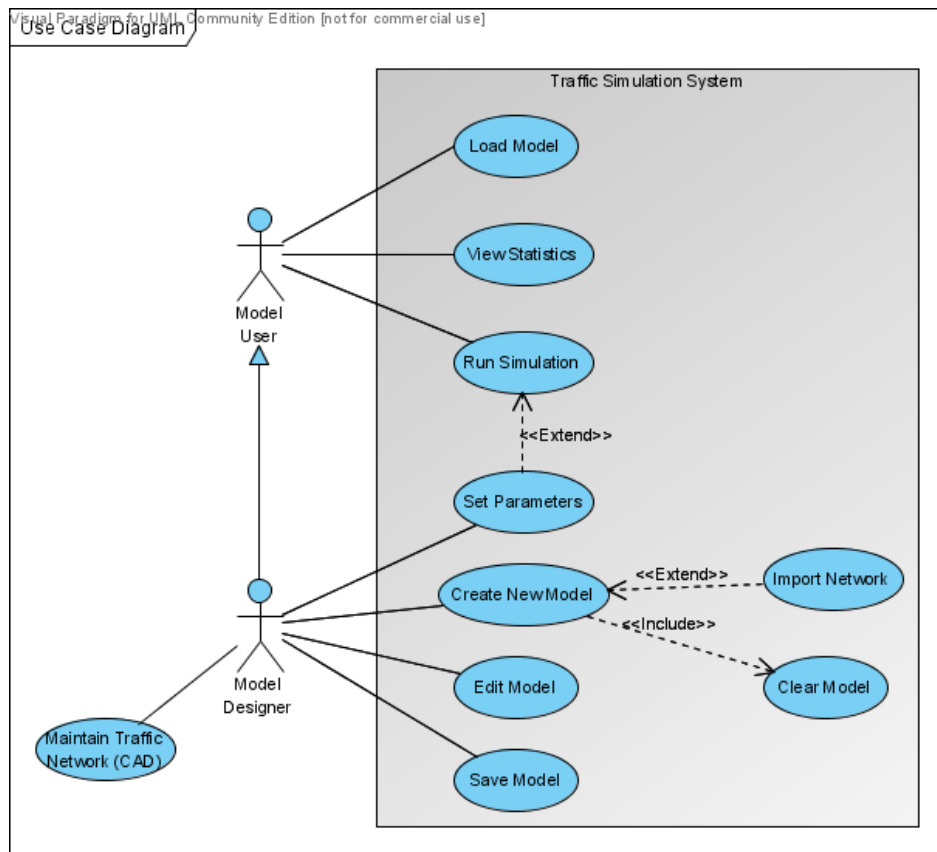


Figure 5.1: Use Case Diagram

Load Model

The user chooses to load an existing model. The system provides a file chooser to locate the existing model. The user finds the file and accepts. The system loads the model from the file chosen, including roads, vehicles and parameters.

Run Simulation

The user optionally alters the simulation parameters (see Set Parameters use case) and chooses to run the simulation. The system starts the simulation and provides visual feedback while it is running. The user chooses to stop the simulation when they have seen enough. The system stops running the simulation and keeps the model in this state, allowing the user to subsequently Run Simulation from where it was stopped.

Set Parameters

The user or developer chooses to alter the simulation parameters. The system displays available parameters to change. The user alters any parameters they wish using graphical interface widgets, and chooses to finish altering parameters. The system stores the new values of any changed parameters.

View Statistics

The user chooses to view output statistics. The system displays all recorded statistics in graphical and text format.

Create New Model

The developer chooses to create a new model. The system runs the Clear Model use case to create a blank model. Optionally, the developer can choose to Import Network. The system is now ready for model editing and parameter setting.

Import Network

The system provides a file chooser. The developer finds an existing traffic network file saved from a CAD package. The system verifies and loads the existing network into the model.

Clear Model

The system restores the model to its default state, i.e. no traffic network, vehicles or other features. It also resets any global model variables and parameters to their default values so that a “Clear” model is always the same regardless of the previous state of the model.

Edit Model

The user chooses from a number of model design tools. The system activates the selected tool. The user selects points on the graphical environment representation. The system updates the model based on the tool and the point selected. The user continues using tools until they are finished.

Save Model

The user chooses to save the model to disk. The system provides a file chooser to determine where to save. The user selects an existing file or types in the name for a new file and accepts. The system converts the model state into serialised form and writes it to the chosen file.

5.1.4 User Requirements

User requirements are high-level functions that the user expects the system to make available. They do not include technical details, but should provide enough information to allow the developer to determine if they have been accomplished. The first five are derived primarily from use case analysis and the others are from the real user requirements researched in section 3.7. UR-9 and UR-10 represents the chosen important/interesting features that this specific project will focus on, based on requirements analysis in section 3.7. UR-11 refers to the specific requirements for modelling the study area in section 2.2.3.

User Requirements

UR-1. The system must provide graphical model creation/editing utilities.

UR-2. The system must allow the user to load/save the current state of the model from/to a file.

- UR-3. The system must allow the user to alter simulation parameters, before and during a simulation run.
- UR-4. The system must provide a visualisation of the current state of the simulation/model, and animate this during simulation runs.
- UR-5. The system must show the simulation statistics graphically using the implemented indicators.
- UR-6. The system must deliver realistic (verified) microscopic and macroscopic traffic behaviour in both urban and motorway scenarios.
- UR-7. The system must be able to simulate a period of at least 30 minutes in simulation time.
- UR-8. The system must run at a speed equal to or greater than real-time.
- UR-9. The system must simulate the following additional features: queue spill back, HOV lanes and co-ordinated traffic signals.
- UR-10. The system must implement the following indicators: travel time, congestion level and AVO (average vehicle occupancy; for evaluating HOV lanes).
- UR-11. The system must be able to model a length of 2-lane road in excess of 1.5km (the length of road of interest in the study area) and be able to simulate the appropriate number of vehicles on this road.

User Requirements Not Fulfilled

One requirement identified is the need to be able to import an existing traffic network from a CAD format. Because no CAD package is available, and this is not central to the development of the simulator, this requirement will be considered outside the scope of the project. Instead, all traffic networks will be created and edited within the simulator in proprietary format. As has been mentioned previously, many features required for real simulators have been purposefully omitted to allow the focus to be on the critical and interesting features, e.g. HOV lanes.

5.1.5 Choice of Tool

To assist in the development of the system, a package will be used that provides agent-based simulation abilities as described in section 4.2.3. The decision was made to use an IDE rather than a toolkit, because all development can be done in a single environment designed specifically for agent-based modelling. Since the IDE is designed for such a purpose, it provides useful tools and its own language geared towards modelling and simulation. This means that time can be spent developing the model without learning how to interface with a toolkit or attempting to program an ABM style using a traditional language.

A number of IDEs are considered in section 4.2.4, of which, NetLogo has been chosen as the tool used for this project. The reasons for this choice are as follows; where a feature relates directly to a requirement, that requirement is noted in brackets. Firstly, it is designed for simulating complex

phenomena, such as traffic, and makes it possible to develop such a model in a logical manner. It provides real-time visualisation of the environment as standard (UR-4), and can simulate any number of agents in that environment, each of which is individually maintained and processed. Local variables allow each agent to have its own behavioural parameters, so agents can behave individually and realistically (UR-6). User interface widgets can be added that allow the user to output (graphs, monitors) and alter (buttons, sliders, drop-down lists) simulation parameters at run-time (UR-3, UR-5). A model can be built in such a way that the user can interact directly with the model (UR-1), and commands exist to allow data to be imported or exported to files (UR-2).

As well as providing features that allow all the requirements to be implemented, NetLogo is also a tool well suited to prototyping. This is because it is fast and easy to create a simplistic model that provides basic functionality, it allows interactive visual UI design, it uses an abstract ABM-specific high-level interpreted language, and it allows the simulation to be run and re-run immediately after code modification. In addition, the system is free to use, allows models to be exported as Java Applets and run in any environment, provides many examples and has a helpful user group.

The tool used in this project for designing UML diagrams is Visual Paradigm for UML 5.2 Community Edition; chosen because it is free to use the community edition, yet provides powerful UML 2.0 features and supports all diagram types.

5.2 Software Design

5.2.1 System Requirements

System requirements are detailed functional and non-functional requirements regarding the technical design of the system and each refers to the implementation of one user requirement. Functional requirements describe the specific functionality of the system and non-functional requirements are constraints on the system that are usually qualitative measurements like quality, performance and cost.

Functional Requirements

- FR-1.1 The system must allow the world to be cleared to begin a new model; this must set all alterable parameters to their defaults, remove vehicles and the traffic network.
- FR-1.2 The system must provide a selection of tools to alter the environment, including add/remove road, vehicle, sink, source and any additional features specified. These must work by acting on the location clicked by the user in the environment visualisation.
- FR-2.1 The system must provide a save button that asks the user to choose a file, then writes the current state of the model into the chosen file.

- FR-2.2 The system must provide a load button that asks the user to choose a file, and loads the saved model.
- FR-3.1 The system must provide graphical widgets to alter all user-settable simulation parameters, including simulation speed/accuracy, number of vehicles, traffic signal timing and percentage of HOV vehicles.
- FR-4.1 The system must utilise the visualisation built-into NetLogo to show the model during editing and running.
- FR-5.1 The system must draw NetLogo graphs for all indicators and also for the time-space diagram used to validate vehicle behaviour.
- FR-6.1 The system must implement a car-following scheme to enable vehicles to adjust their speed according to the situation and their parameters.
- FR-6.2 The system must implement a lane-changing scheme allowing vehicles to change lanes when they see it as beneficial.
- FR-6.3 The system must allow each vehicle to have its own behavioural parameters, including max speed, max acceleration and max deceleration.
- FR-6.4 The system must be valid at a microscopic level, i.e. individual vehicles must behave in a realistic manner.
- FR-6.5 The system must be valid at a macroscopic level, i.e. the traffic stream must coincide with realistic data found in traffic studies.
- FR-7.1 The system must show the current simulation time in hours, minutes and seconds.
- FR-7.2 The system must be able to sustain a valid simulation run at least until the simulation time is 30 minutes.
- FR-8.1 The system must show the current simulation speed, as compared to real-time.
- FR-9.1 The system must simulate queue spill back by making vehicles able to stop at a junction when the new road is full.
- FR-9.2 The system must differentiate between standard lanes and HOV lanes. Vehicles must have a parameter holding the number of people in the vehicle, and vehicles must abide by the rules of the HOV lane, i.e. not use it unless there are two or more people.

FR-9.3 The system must enable the placing of traffic signal agents onto roads. These must have a custom-set green-time in each direction, plus a preset yellow-time for safety. It must be possible to coordinate separate signal blocks. Vehicles must abide by the traffic signals.

FR-10.1 Travel time must be tracked by recording the time (in seconds) at which each vehicle enters the simulation, and comparing this with the time at which they exit.

FR-10.2 Congestion level must be recorded by use of a flow counter agent that calculates hourly flow data based on traffic going past it.

FR-10.3 AVO must be recorded by use of a counter agent that checks the occupancy of each vehicle that goes past it and reports the average.

Non-Functional Requirements

NFR-5.1 Distinguished features should be shown on the graph in different colours so they can be seen apart, including HOV vehicles vs. other vehicles, stopped vehicles vs. moving vehicles.

NFR-8.1 The system should sustain a simulation speed of at least one (real-time) when running a reasonable model such as that of the local study area.

NFR-8.2 The system should provide an option to force it to run in real-time to allow the visualisation to be easily viewed.

NFR-11.1 The system should be able to model a length of 1.5km through adjusting the NetLogo world size accordingly.

NFR-11.2 The system should be able to simulate over 300 vehicles over the 1.5km length of road.

5.2.2 Using NetLogo

NetLogo, introduced in section 4.2.4, is an integrated environment for agent-based modelling of complex phenomena. There are two types of programmable agent in NetLogo: patches and turtles. Patches have a fixed location, are arranged in a grid, and can be used to represent discrete sections of an environment. They are appropriate for modelling the road sections, which will then be restricted to a grid. Turtles are movable agents that exist in a continuous world sectioned into patches, and will be used for vehicles and other objects. The world can optionally wrap in either axis, forming a torus or cylinder, easily allowing infinite lengths of road. Agents have several built-in properties, such as x and y coordinate, colour, label and heading for turtles. Agents can be given custom properties, allowing them to have their own behavioural parameters. Turtle agents can be classified into ‘breeds’, where each breed has a separate set of custom properties and can be referenced as a group. This allows vehicles to be differentiated from, for example, traffic lights.

Functionality is given to agents in the form of procedures, which run in the context of an individual agent. Properties of that agent (and of the patch under it, if a turtle) are directly available in the procedure. Procedures are programmed in an extended Logo language that provides most programming constructs including conditionals, loops, recursion, and many NetLogo-specific constructs. NetLogo implements dynamic weak typing, meaning data types are automatically assigned and converted at runtime, and supports numbers, strings, booleans, lists (containing elements of any type), agents and agentsets (collections of agents). NetLogo imposes no restrictions on procedures – they can always be run by any agent. If a procedure attempts to use a property that the agent in context does not support, an error occurs at runtime. Care must be taken to ensure each procedure is always called by the correct type of agent.

By convention, most models have ‘setup’ and ‘go’ buttons. The ‘setup’ button calls a procedure to reset the model to its initial state, and the ‘go’ button repeats a procedure that carries out all actions for each time step. A ‘step’ button can be added to perform only one run of the ‘go’ procedure, advancing the simulation by one time-step. NetLogo manages all visual aspects of the simulation, and automatically updates the visualisation after every time-step. Turtles can be given shapes, sizes and colours, which govern their appearance in the visualisation. Agents communicate by ‘asking’ each other to perform actions, and an agent always performs said actions.

Scheduling in NetLogo works by allowing agents to take turns in completing a minimal unit of work. Depending on the commands run by each agent, agents can finish the commands at different times. Splitting commands into different ‘ask’ blocks ensures all agents have completed the first block before any start on the second block. This will be used to ensure all vehicles have completed lane-changing before any begin vehicle-following. The ‘without-interruption’ command can be used to force each agent to run the whole block of commands without switching context.

NetLogo models are not written in an object-oriented manner, since procedures apply to any agents who run them. However, it is possible to design and write code in an OO style, by treating breeds as subclasses of turtle and making use of breed properties. Procedures and properties can be ordered so that all code relating to each breed is kept together, separated perhaps by comments. Code convention in NetLogo is to separate property and procedure names with the minus symbol, so this will be adhered to. In addition, global and class variables will be uppercase to distinguish them from instance variables because NetLogo makes no distinction. Some aspects of standard OO programming are lost with NetLogo, e.g. you cannot have subclasses of breeds or patches, although this is probably a good point because models would get too complex.

5.2.3 Class Diagram

NetLogo provides many inbuilt turtle and patch variables; only the ones which are relevant to the project are shown on the diagram. All properties and procedure are public because any agent can query any other, therefore no visibility will be shown. Although NetLogo does not provide an object-oriented environment, conceptually the system will be designed using OO techniques. Although all the procedures could be called by any agent, each procedure is specific to one type or breed of agent. On this diagram, yellow classes are those provided by NetLogo, and blue classes represent those elements added for this project.

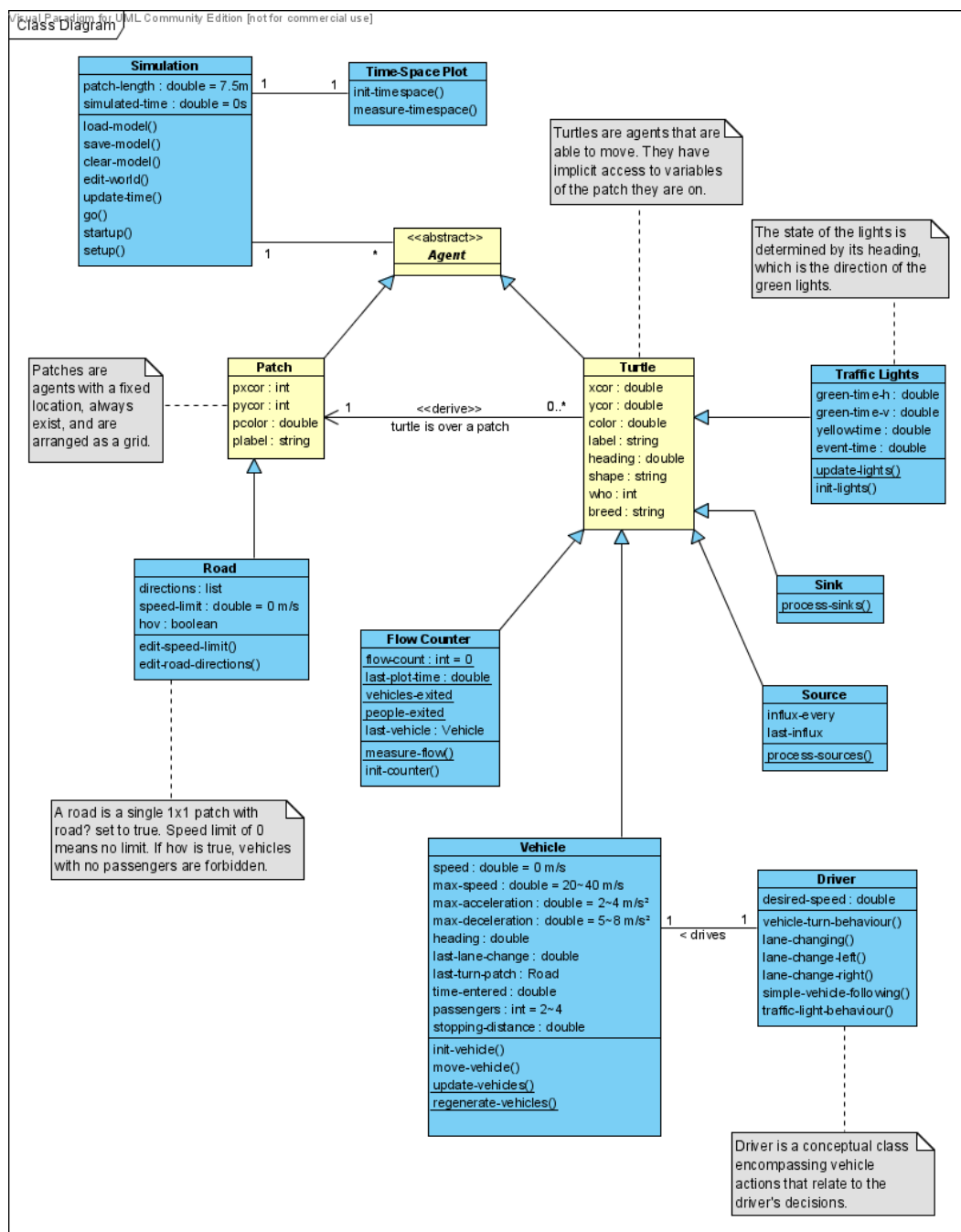


Figure 5.2: System Class Diagram

In contrast to most traffic simulators, which represent the traffic network using links and nodes (see section 3.5.3), this system represents the road on a discrete grid of cells. This is because NetLogo is designed to represent the environment on which agents roam with patches. The disadvantages of this approach are that roads must be either vertical or horizontal, and complex junctions are impossible to model. However, this still allows many interesting scenarios to be modelled. Nodes are not explicitly defined; a junction is merely a road with roads leading to it from many directions.

The Vehicle/Driver model is straightforward. The core parameters listed in 3.5.1 are all implemented, allowing accurate vehicle behaviour. To simplify the vehicle-following and lane-changing algorithms, only one driver-specific parameter is implemented, and it is altered by the various driver behaviour methods. Each driver behaviour method is called in order of increasing priority so that the most important desired-speed is the one used to update the vehicle.

5.2.4 Interaction Diagram

Sequence diagrams are a type of interaction diagram that show the flow of messages through a system as a particular function is being executed. The following is a sequence diagram describing the sequence of events that occur during a simulation run. It describes a single time-step, and during normal use, this sequence would be repeated for the duration of the simulation.

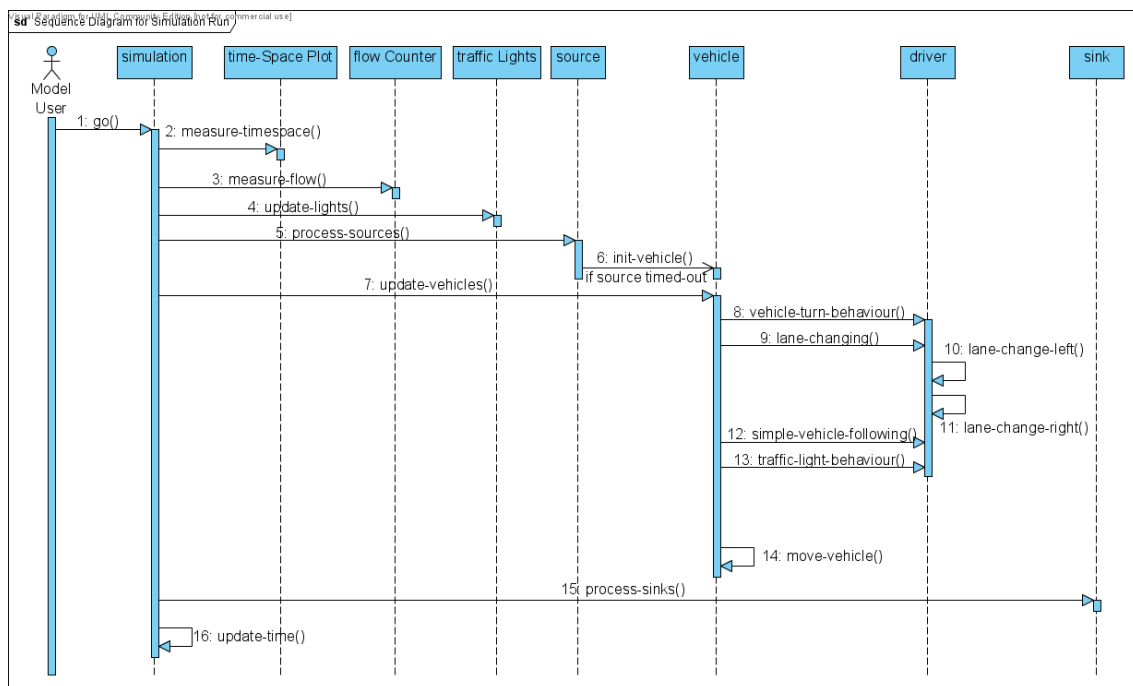


Figure 5.3: Sequence Diagram of "go" procedure

5.3 Test Plan

The testing methodology described in section 5.1.2 outlines three levels of testing: for runtime errors, realistic behaviour of individuals (microscopic) and valid overall behaviour (macroscopic). The following plans for testing will be carried out at each stage of implementation to ensure the system is performing correctly. The system will also be tested on the problem, i.e. analysing HOV lanes on the traffic network specified in section 2.2.3, to learn whether it is useful in a real-world problem.

5.3.1 Error Testing

Tests will be completed to ensure that each requirement is implemented correctly; these requirements will be ticked off as they are implemented and tested. Various situations will be tested to ensure that no combination of events can cause a runtime error. These tests will be ongoing and will be resolved as and when they occur, therefore they do not require explicit test plans. Some example test situations are:

- Placing a vehicle not on the road
- Running the simulation with no vehicles
- Running the simulation with no roads
- Attempt to generate more vehicles than there are roads
- Enter invalid input into any input box

5.3.2 Microscopic Validation

This involves analysing the behaviour of individual vehicles under varying conditions, and is required due to FR-6.4. In addition to watching the visualisation, a space-time diagram (introduced in section 2.3.2) can be used to analyse the behaviour of vehicles over time. This feature will be built into the system to enable the movement of vehicles to be tracked and verified. In particular, the following tests will be run to validate certain microscopic behaviours:

<i>Test</i>	<i>Test Conditions</i>	<i>Expected Result</i>
T-1	Single-lane road with one stopped vehicle.	Vehicles should slow as they approach the stopped vehicle and stop moving when exactly one block behind.
T-2	Single-lane repeating road with few vehicles.	Vehicles should free-flow continuously.
T-3	Single-lane repeating road with many vehicles.	Vehicles should form a traffic jam that moves backwards in the space-time diagram.

<i>Test</i>	<i>Test Conditions</i>	<i>Expected Result</i>
T-4	Two-lane repeating road with one stopped vehicle.	Vehicles should move out a lane to pass the stopped vehicle, and then optionally move back a lane.
T-5	Single-lane repeating road with one traffic light.	Vehicles should stop without moving past the light when it is red. Vehicles behind should stop in queue.
T-6	Single-lane road that turns left/right then traffic light.	Vehicles should turn and obey the light. If red, vehicles should queue back around the corner.
T-7	Two-lane repeating with HOV lane.	Vehicles with no passengers should stay out of the HOV lane.
T-8	Single-lane road with left junction.	Most vehicles should drive on but some should turn into the side road.

Table 5.1: Microscopic Validation Test Plan

5.3.3 Macroscopic Validation

High level testing ensures that the output from the system is useful and valid. If the system is not calibrated properly, it could invalidate all output. Nagel [28] suggests that a suitable way to verify a traffic simulator's macroscopic properties is to generate and analyse the fundamental diagrams such as flow-density diagrams. In order to validate this, flow counters will be placed on the road to count the number of vehicles passing over them. The data from these counters can be plotted on a flow-density diagram (section 2.3.2).

NetLogo provides a feature called BehaviourSpace, which allows multiple simulation runs to be completed while varying simulation parameters. For this test, a repeating length of road will be modelled, and BehaviourSpace will be used to vary the density from one vehicle to the maximum density. The resulting flow-density diagram should resemble those shown in section 2.3.2. Under optimal conditions, traffic flow should reach 2,000 vehicles per hour, per lane at around 40 vehicles per km, per lane (see section 2.3.1). The flow counters will calculate the hourly flow every minute of simulated time so that the test does not take as long as it would in real-life, and each run will last for two minutes of simulated time, to double-check the results.

Chapter 6: Implementation and Evaluation

6.1 Phase One

6.1.1 Phase Implementation Overview

Following the prototyping methodology, the first phase of implementation was to program a simplistic model and simulation, with only the core features realised; primarily vehicle-following behaviours. The following requirements were fully or partially implemented in the first phase:

<i>Requirement</i>	<i>Description of Implemented Feature (✓ means requirement completed)</i>
FR-1.1 ✓	“New Model” button that clears all roads and turtles, and resets parameters to their default values.
FR-1.2	Selection of two editing tools; road and vehicle. Roads can be drawn onto the grid of patches and vehicles can be placed.
FR-2.1 ✓	“Save Model” button was simple in NetLogo; one command does it all.
FR-2.2 ✓	“Load Model” button is as simple as the Save button.
FR-3.1	“Time Slice” slider allows one to adjust the speed and accuracy of the simulation. “No of Vehicles” slider allows one to select how many vehicles are generated.
FR-4.1 ✓	The visualisation in NetLogo is implicit – there is no work to do because, by definition, the model includes the animated 2D and 3D visualisation.
FR-5.1	Flow-Density graph has been implemented; it uses flow counter agents to plot one point per simulated minute. Time-Space graph has been implemented; this draws a line for each vehicle, plotting its x coordinate with time.
FR-6.1 ✓	Vehicle-following behaviour was challenging to make realistic, but works well.
FR-6.3 ✓	Vehicles have individual parameters for max-speed, max-acceleration, max-deceleration and desired-speed; these are used in the vehicle-following behaviour.
FR-7.1 ✓	“Simulation Time” monitor shows a formatted time string in hours, mins and secs.
FR-8.1 ✓	“x Realtime” monitor shows how many times real-time speed is being achieved.
FR-10.2 ✓	Flow counters have been implemented and output their hourly data to the flow-density graph every minute of simulated time.
NFR-5.1	Time-Space graph shows vehicles in a jam as red rather than usual black.
NFR-8.2 ✓	“Realtime” switch makes the simulation adjust the time slice to keep it in real-time.

Table 6.1: Requirements Implemented in Phase One

6.1.2 Phase Implementation Details and Issues

The main development issues in this phase were to setup the simulation using real-world time and space figures, to represent vehicles and roads, and to implement vehicle-following behaviour.

Time/Space Management

Similar to CA simulators, the system is designed so that each vehicle occupies exactly one patch of road when it is stopped in a queue. It has been shown that a reasonable value for cell length in CA simulators is 7.5m, so this value will be used as the patch length [28]. This allows vehicle speeds to be calculated in real units (meters), and when updating vehicles they are simply moved by $(\text{speed}/7.5)$ patches. Time also plays a role; smaller time-slices will result in smaller, more accurate movement steps being made by vehicles. This results in vehicles being moved by $(\text{speed} / 7.5 * \text{time-slice})$ at each tick.

Vehicle Following

The vehicle following algorithm implemented in this phase is based on the anti-collision concept and was inspired by the Gipps model and fuzzy logic ideas (see section 3.6.1):

```

IF (any vehicles in front within current stopping distance)
    Slow down (using max deceleration)
ELSE
    Speed up (using max acceleration, restricted by maximum speed)

```

This algorithm was intended to model the core features of vehicle interaction, by making drivers drive in such a way that they would always be able to stop in time if the vehicle in front stopped. Initial testing (below) has shown that this algorithm results in reasonably realistic vehicle following behaviour. It appears to give drivers the ability to follow the well-known two-second rule, where drivers attempt to keep at least two seconds worth of distance between vehicles. It also makes vehicles slow-to-start, i.e. they do not move until the vehicle ahead is a certain distance away. More detail on the vehicle-following model, including other options evaluated, is in appendix B-1.

6.1.3 Phase Testing

This phase has made possible a vehicle-following simulation with a one-lane road. To verify the behaviours, the following microscopic tests from section 5.3.2 are run: T-1, T-2 and T-3. The behaviour was as expected and these tests have passed; the resulting time-space diagrams used to verify these tests are shown, in order, below.

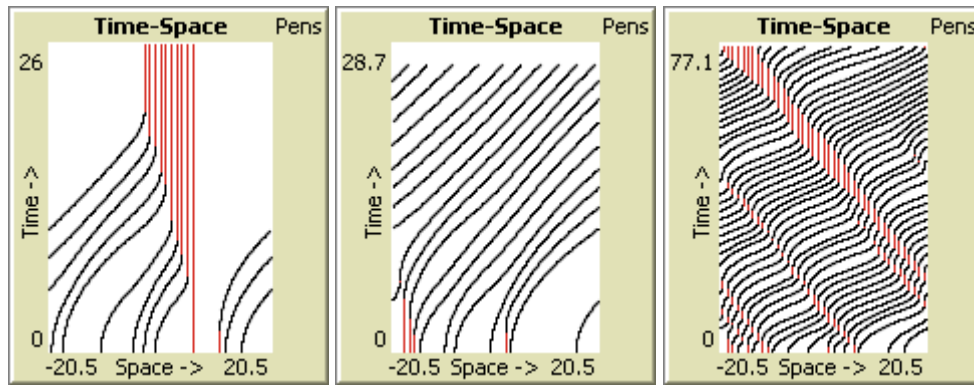


Figure 6.1: Phase One Microscopic Tests T-1, T-2 and T-3

To test the model for its macroscopic validity, the test described in section 5.3.3 was used. The time slice chosen was 0.2 seconds, which provides a good level of accuracy and speed. Number of vehicles was varied from 2 to 30, governed by the length of the road, which was long enough for 30 vehicles. Density was measured as the percentage of road occupied by a vehicle. The resulting graph shows that traffic flow is being modelled realistically.

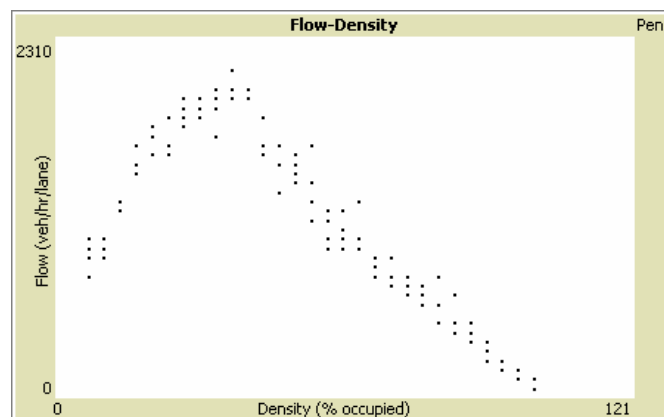


Figure 6.2: Phase One Flow-density Test

The maximum flow recorded was 2090 vehicles per hour per lane, which was recorded at 37% density (about 49 vehicles per km). This matches the maximum flow recorded in traffic studies of around 2000 vehicles per hour per lane, and is close to the critical density of 40 vehicles per km. The shape of the flow-density graph follows the same curve as of those in the literature and traffic studies (see 2.3.2), although there appears to be no synchronised traffic state. This is possibly because no comfort-factor has been implemented.

6.2 Phase Two

6.2.1 Phase Implementation Overview

The second phase brought necessary features required to simulate urban and motorway traffic. The following requirements were implemented in the second phase:

<i>Requirement</i>	<i>Description of Implemented Feature (✓ means requirement completed)</i>
FR-1.2	Sinks, Sources and Traffic Lights can now be added using the model editing tool. Roads can now be given one or more directions that control which way vehicles travel on the road. Roads can also be given a speed limit.
FR-3.1	Traffic signal timings are prompted upon placing a new light. To alter timings, the agent inspector can be used, or simply remove and add the light.
FR-5.1	Travel time is now shown on a graph against simulation time. It is calculated whenever a vehicle enters or leaves the simulation.
FR-6.2 ✓	Lane-changing behaviour was the most difficult aspect to make at-all realistic, even with many pseudo-code algorithms available. More details below.
FR-9.1 ✓	Because roads now have directions, they can turn for corners or junctions. Vehicles follow each other and stop if there is a queue. Likewise with traffic lights.
FR-9.3 ✓	Traffic lights force vehicles to stop when they are red in their direction. They can be coordinated by placing them at the same time and setting the green-times wisely.
FR-10.1 ✓	Travel time is now recorded by new vehicles and sinks, and graphed.

Table 6.2: Requirements Implemented in Phase Two

6.2.2 Phase Implementation Details

This phase saw the addition of the following:

Sources and Sinks

A message box is used to input the desired interval for sources, which then generate a vehicle with the same speed as the vehicle in front every interval. Sinks simply remove any vehicles that cross them from the simulation.

Traffic Lights

A message box is used to input the desired green-times in both horizontal and vertical directions. The light simply faces the direction in which the lights are red, and vehicles look for lights then compare their direction to the direction of the red lights.

Roads with direction

Roads are given a list of directions in which vehicles are allowed to leave each road patch. If a road has one direction, vehicles are obliged to turn if they are heading a different direction. If a road has more than one, vehicles turn with a default chance of 1 in 5.

Lane Changing

The lane changing behaviour implemented is based on the incentive-safety idea (see section 3.6.2), and is expressed below.

```

IF (current leader is slowing you down)
    IF (safe distance ahead and behind on right lane)
        MOVE RIGHT
    ELSE
        IF (safe distance ahead and behind on left lane)
            MOVE LEFT

```

The safety distance depends on the direction moving – when moving left it is probable that the car is travelling faster than the car behind, so a short distance is safe. When moving right the car checks the speed of the vehicle behind and calculates whether it is safe. To minimise unnecessary lane changes, the vehicles evaluate the situation every one second, and change at most every 5 seconds.

6.2.3 Phase Testing

This phase has added various features requiring testing, most importantly lane-changing, which is the most complex behaviour in the system. To test the lane-changing behaviour, the same macroscopic test can be run but with two lanes. This will verify both the lane-changing and vehicle-following behaviours and ensure they work together. The flow-density graph is smoother than the first one, and still exhibits plausible qualities. Flow reaches 2,000 vehicles per hour at a density of roughly 30%, which is 40 vehicles per km, and matches the expected density and flow.

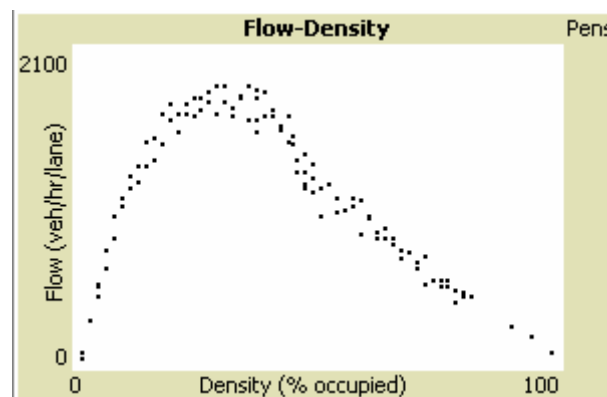


Figure 6.3: Phase Two Flow-density Test

From the microscopic tests, T-4, T-5, T-6 and T-8 apply to this phase. These tests all perform as expected, so the behaviours introduced can be considered valid. The time-space diagrams used to verify T-4 and T-5 are shown below; T-6 and T-8 were validated by watching the visualisation, because the time-space diagram can only show movement in one direction. In T-5, sometimes a vehicle had to slow or stop behind the stopped vehicle temporarily because it was not safe to move into the other lane.

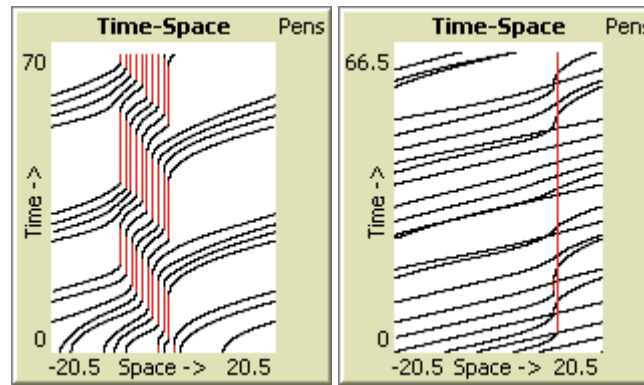


Figure 6.4: Phase Two Microscopic Tests T-4 and T-5

6.3 Phase Three

6.3.1 Phase Implementation Overview

The third phase introduced the advanced features required to simulate HOV lanes. The following requirements were implemented in this phase:

Requirement	Description of Implemented Feature (✓ means requirement completed)
FR-1.2 ✓	HOV lanes are now supported by the model editing tool.
FR-3.1 ✓	“% HOV” slider sets the percentage of new vehicles that have 2+ people.
FR-5.1 ✓	There is now a graph showing AVO, updated every minute.
FR-9.2 ✓	Each road now has a HOV boolean to determine whether it is a HOV lane or not. Vehicles have a number of passengers so they know whether to use the HOV lane.
FR-10.3 ✓	AVO is recorded by existing flow counters by checking the number of passengers in each vehicle passing over it.
NFR-5.1 ✓	The travel time graph now shows HOV vehicles in red to compare travel times.

Table 6.3: Requirements Implemented in Phase Three

6.3.2 Phase Implementation Details

With the introduction of HOV lanes, a few existing features had to be tweaked. First, it was assumed that the HOV lane would be to the left of the normal lane, which made it more straightforward to alter the vehicle behaviours. Vehicles now check whether they are on a HOV lane, and if so, and they have no passengers, they will attempt to move right until it is safe. Vehicles moving left check whether they are allowed in the lane before moving.

6.3.3 Phase Testing

From the microscopic tests, T-7 is applicable to this phase. This works reasonably well although there are issues – if there is no space to move in or out of the HOV lane then the vehicle continues in the wrong lane. However, if vehicles are introduced to the non-HOV lane then they will only

enter it if allowed. This was tested by watching the visualisation, because the time-space graph cannot differentiate between lanes. At the end of the project, an additional macroscopic flow test was run to see if the hysteresis effect described in section 2.3.2 could be modelled. Details of this can be found in appendix B-4.

6.4 Testing on the Problem

This section demonstrates the system by performing a walkthrough of the major steps required to model and simulate the traffic network described in section 2.2.3. This will also allow the remaining non-functional requirements to be checked by implementing and using a large model.

6.4.1 Modelling the Network

The first step to modelling a real network is to gain the length of the links and location of the nodes. By using a map, the lengths of each link in question were measured in metres. The section of the Avon Ring Road modelled is shown below, with distances. In the model, angles are ignored and the road is turned into a grid pattern. The green and blue lanes will be modelled as both HOV and normal lanes. To simplify the problem, only westbound roads will be modelled; it is assumed that eastbound traffic does not affect westbound traffic.

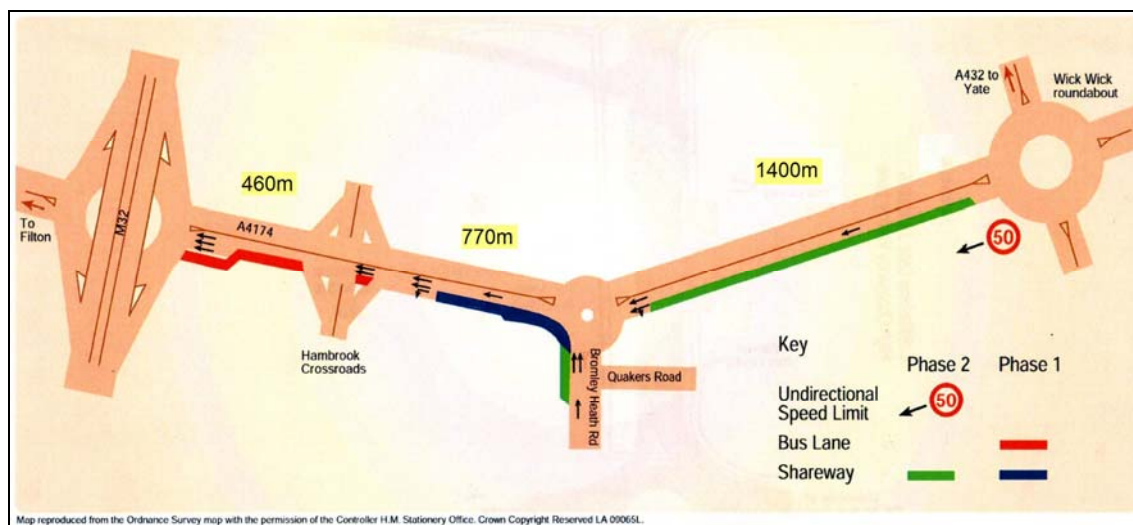


Figure 6.5: Modelled Section of Ring Road. Source: [44]

To make it possible to model this network, the real world units must be converted into patches. Each patch is 7.5m long, so dividing each length by 7.5 gives the length in patches for the road. Calculation shows that the model will require a world about 400 patches wide; the height has to be enough to model part of the side roads, i.e. 40 patches. The world dimensions are set using NetLogo's World & View Edit button, shown below.

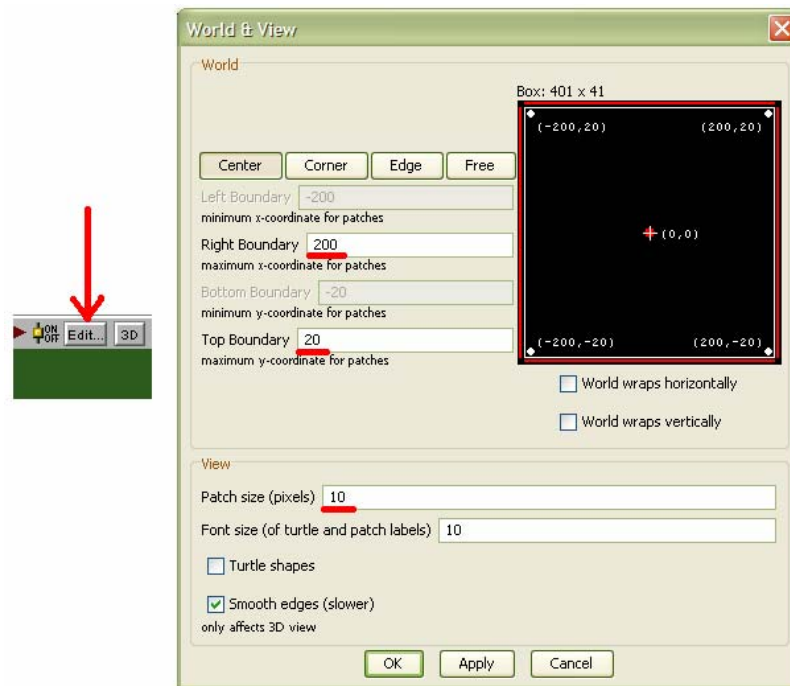


Figure 6.6: NetLogo World & View Screenshot

In the 'Center' mode, the total width and height is double that which is entered into the box. The patch size can be set to make the visualisation smaller or larger. At least 10 pixels are required to create a model. Horizontal wrapping can optionally be set to build a repeating road. Once the world is the correct size, use the model editing tools to create the network:

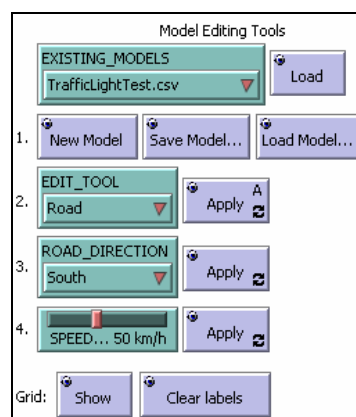


Figure 6.7: Model Editing Tools Screenshot

First, the New Model button clears the model. Then the Edit Tool should be set to Road, and click Apply next to it. Click the Show button at the bottom to show labels on the patches to help make the road the right length. Then the road is simply drawn onto the environment. The Road Direction tool is used to set which direction(s) each road is used for, and can be used in conjunction with the Edit tool. HOV roads are shown in red.

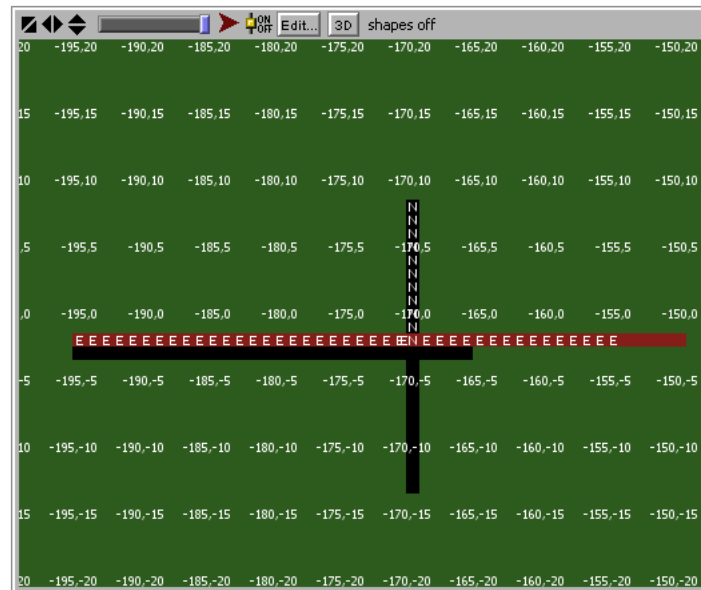


Figure 6.8: Model Creation Screenshot

Once the roads are in place, use the other edit tools to add speed limits, sinks, sources and traffic lights. Clear the labels for clarity then click once on a road to apply them; click again to remove them. Sources and traffic lights will present timing options, as in the example below.

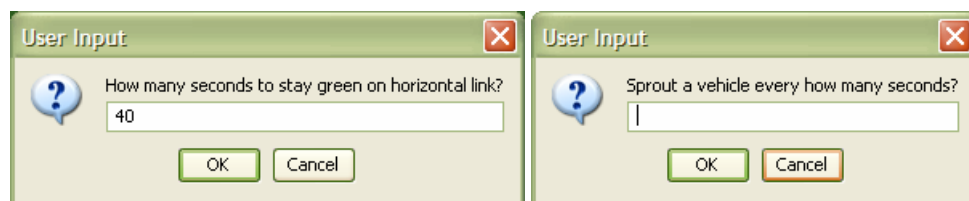


Figure 6.9: User Input Screenshots

When the network is complete, it can be tested. Start by choosing a number of vehicles (choose zero if using Sources for vehicle influx), then press Regen to remove all vehicles and make the number selected. Then press Setup to reset the simulation run; this default all runtime parameters and clears the charts. Then use Go to start the simulation. Numbers above traffic lights correspond to the number of seconds until the lights change. The leftmost coloured disc shows the light for a section of road. The Journey Times chart will always show current journey times from sources to sinks, but for the flow-density or AVO charts, at least one flow counter is required – place one on each lane before sinks. The 3D button can show a 3D controllable visualisation as follows:

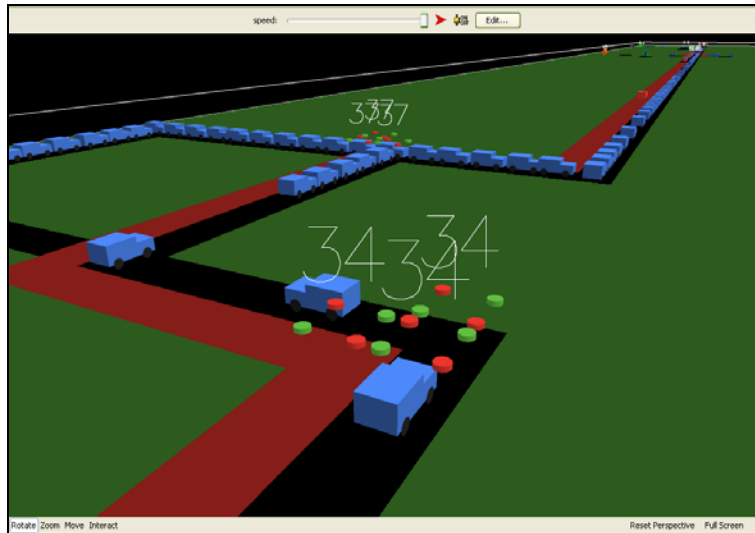


Figure 6.10: 3D Visualisation Screenshot

6.4.2 Simulating the Network

Two simulations were run on this network: one with the HOV lane and one without. The output from the simulation is interesting; the following is a top-down view of the model constructed (see the start of the previous section for a map of the network).

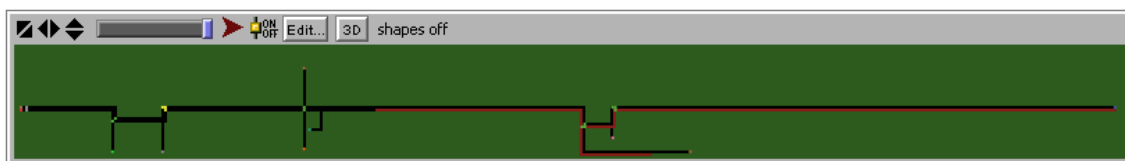


Figure 6.11: Model Overview Screenshot

Simulation Parameters

- Real traffic signal timings taken from fieldwork investigation.
- 25% of vehicles have more than one person (from data in Appendix A-1).
- Time slice of 0.15 for good accuracy.
- The main source (the right hand side) creates a vehicle every 2 seconds. This only needs to be fast enough to cause congestion.
- The side sources create a vehicle every 10 seconds so as not to skew the main results.
- The source very near the left side creates a vehicle every 5 seconds but does not affect the results greatly because it is so close to the end.
- The simulation was run for one hour of simulated time.

Results on Journey Times

Predictably, both results show a similar shape in journey time for those vehicles who enter near the finish – these are the two lines at the bottom. With the normal road all vehicles are treated equally,

and the queues reach up to 40 minutes after an hour of simulation. The normal road does have a selection of vehicles who manage to lower their journey times; this must be due to the secondary sources. The HOV chart is interesting, as the driver-only vehicles reach the same peak of 40 minutes but all the HOV users maintain a maximum journey time of under 10 minutes. Also it manages to limit the journey time of all vehicles from the first junction from the right to 16 minutes. Based on these results, the HOV lane appears to be successful.

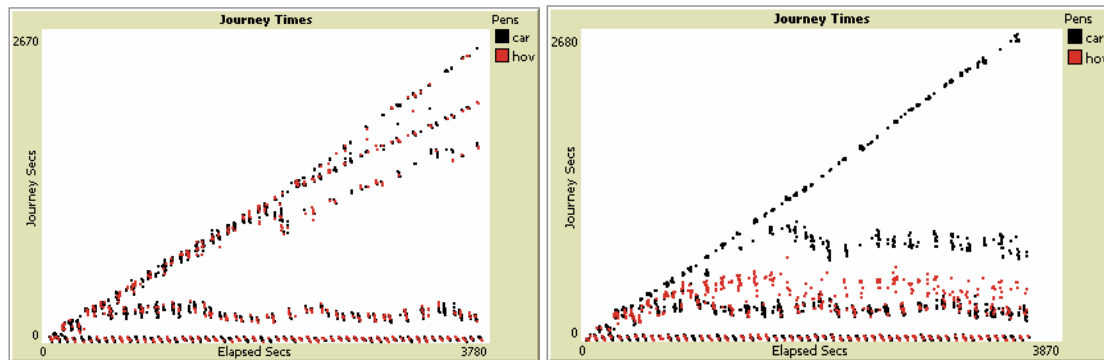


Figure 6.12: Journey Times – Normal vs. HOV

Results on Flow and Density

Now to compare the total traffic flow and density; both go through a similar build up of flow as the density picks up, but HOV settles on around 39% density whereas normal fills up to 63%. Interestingly, even though there are more vehicles on the overall road during the normal test, the flow rate at the sinks is approximately the same as that of the HOV test. This further builds confidence in HOV lanes; as they can sustain a good overall flow without as much dense traffic.

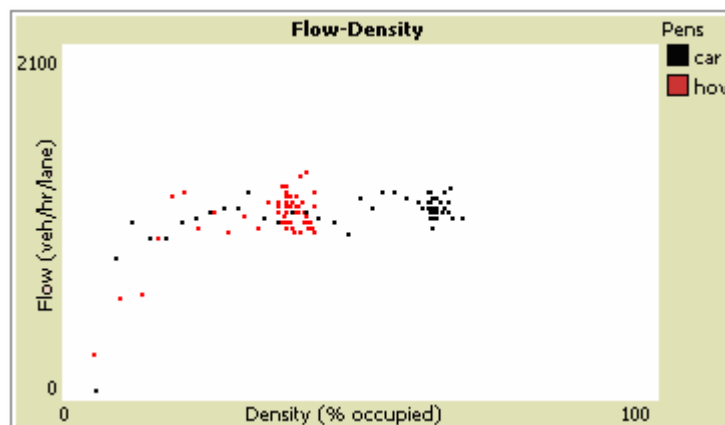


Figure 6.13: Flow-Density Results

Results on AVO

The last comparison is with the AVO results. The normal road had an AVO of 1.38, and the HOV version had an AVO of 1.44, an increase of 0.06, or 4%. It is hard to say whether this is justified or not, as the AVO data was not very concise, as shows the graph below. This is perhaps because the

data was formed on a minute-by-minute basis; there may not be enough data in one minute to generate a meaningful chart. If this is the case, then the increase of 4% over an hour could be justified.

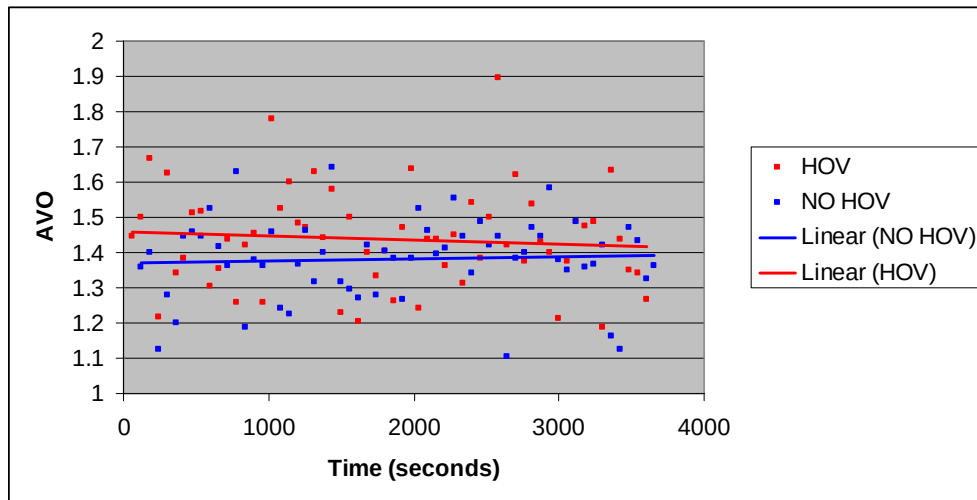


Figure 6.14: AVO Results

6.4.3 Interpreting the Results

The aim of the HOV project is to give an advantage to car sharers and bus users. The statistics from South Gloucestershire Council and the results of the simulation agree that journey times are significantly reduced by use of the HOV lane. The council claim to have reduced the journey time for all vehicles by 10 minutes, with HOV users saving the most time. The simulation suggests that the HOV users save up to 30 minutes but the rest of the traffic does not get a reduction in journey time.

The statistics from the Council stated that the traffic flows have increased by 36%, the literature states that flows should decrease by 10%, yet the simulation suggests that traffic flows remain the same. This needs further investigation. The council state that AVO has increased by approximately 7% and the simulation suggests 4%, which is reasonable. More parameters need to be tested in a large variety of situations in order to claim that the simulation results are conclusive, but they seem to be on the right track.

6.4.4 Performance Testing

Regarding the non-functional requirements that have not yet been verified, NFR-11.1 is shown to be valid by the fact that the system allowed the 1.5km length of road to be modelled in this section. NFR-11.2 was tested during the running of the simulation; the maximum number of vehicles being simulated successfully was approximately 650, well above the requirement for 300 vehicles. To test NFR-8.1, the 'x realtime' monitor was checked at various stages of simulation. Execution speed

depends on the computer speed, the time slice, the number of vehicles, and the network design. With the large network discussed above and a time slice of 0.15, the execution speed is below the required standard: 350 vehicles runs at 0.3 times realtime, and 650 vehicles runs at 0.12 times realtime. However, this is using a Pentium III 800MHz PC; with a new machine, 350 vehicles would likely run in realtime.

Chapter 7: Conclusions and Future Work

In this project, a traffic simulator has been developed using NetLogo, an integrated agent-based modelling environment. Features and indicators that have been implemented include vehicle-following, lane-changing, sources, sinks, traffic lights, HOV lanes, traffic flow, journey times and vehicle occupancy. The simulator has been calibrated and validated with known data from traffic studies, and testing has shown that the simulator accurately models traffic flow in a variety of urban and motorway conditions. It has been successfully applied to modelling a large westbound section of the Avon Ring Road to analyse the effectiveness of HOV (High Occupancy Vehicle) lanes.

When applied to the problem of analysing congestion, simulation proves to be a useful tool. Promising results are revealed when the Avon Ring Road model is run with and without the HOV lane, suggesting that travel times are reduced substantially for HOV users and not affected for others. According to the results of the simulation, HOV lanes are certainly effective in this case. The only calibration data not properly considered was vehicle input rate, although this may not be important as long as vehicles enter at identical rates in both tests. Investigation into the effects of other congestion reducing schemes and in other study areas is left for future work, as is the simulation of a larger area of the Ring Road, including eastbound traffic.

The requirements detailed in Chapter 5 were successfully met by the system implementation. The most challenging requirements to address were car-following and lane-changing behaviours. This was expected due to the amount of attention they receive in the literature. Resulting vehicle behaviour is mostly realistic especially at macroscopic level; however, it could be improved through further analysis. Driver characteristics were not modelled; however, vehicles do have individual maximum speed/acceleration resulting in a degree of perceived driver characteristics. Future work on this project includes further calibrating inputs/behaviours, verifying simulation output and implementing more features/indicators. The major applicable feature not implemented was public transport, in particular buses. Due to their high use of HOV lanes and large capacity, they would largely affect vehicle occupancy figures. Other important features to consider are incidents, pedestrians and adaptive traffic signals, among many more.

The agent approach to traffic simulation is well justified, since vehicles and traffic control features can be represented directly as intelligent agents. It makes the problem easier to tackle by allowing the developer to focus on modelling individual agent behaviour. Agent-based modelling allows a traffic simulation to be easily implemented using agent techniques. NetLogo was a key tool, which provided extensive functionality while being highly learnable. It is highly recommended as a first step into the agent-based modelling concept, and made development time extremely productive.

Disadvantages of NetLogo include performance limitations: with a large number of agents, execution speed drops considerably. NetLogo is not designed for extensive simulations, and these can be rebuilt using an ABM toolkit/framework for improved performance. It was not possible to model networks using traditional nodes and links, because patches are intended to represent the environment and agents cannot act as links. The next release is scheduled to contain new agent link constructs, which would enable networks to be modelled in this way thus allowing non-grid traffic networks. The interface editor is impressive, but as it must exist on one panel, groups of commands cannot be hidden or placed in separate windows without splitting the model into two parts. It is also not suitable for 2D visualisation of large world sizes; the entire world is always shown in the visualisation window, which must be made extremely large to enable model editing. In this case, the interface panel must be scrolled and all buttons are lost when not in the visible area.

A great deal of knowledge was gained about the specifics of traffic simulation and agent-based modelling during the course of the project, both from research literature and from personal experience during development. The aims and objectives of the project set out in the introduction have been met, and this report accurately represents the steps taken to achieve the objectives. In conclusion, this project can be considered a success based on the reasons outlined above.

Bibliography

- [1] ALGERS, S. et al., 1997. Review of Micro-Simulation Models. SMARTTEST Deliverable D3, Institute for Transportation Studies, University of Leeds, Leeds, UK, Aug. 1997.
- [2] ALGERS, S., et al., 1997. Review of Micro-Simulation Models Appendices A, B and C. SMARTTEST Deliverable D3, Institute for Transportation Studies, University of Leeds, Leeds, UK, June. 1997.
- [3] ALGERS, S., et al., 1999. Best Practice Manual. SMARTTEST Deliverable D8, Institute for Transportation Studies, University of Leeds, Leeds, UK, May. 1999.
- [4] AL-SHIHABI, T. and MOURANT, R.R., 2001. A framework for modelling human-like driving behaviours for autonomous vehicles in driving simulators, *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, 2001, ACM Press pp286-291.
- [5] APAS, 1995. APAS Roads 2: Assessment of Road Transport Models and System Architectures, European Commission Directorate General for Transport, April 1995.
- [6] BALL, P., 2004. *Critical Mass*. United Kingdom: Arrow Books.
- [7] BALMER, M. et al., 2004. Towards truly agent-based traffic and mobility simulations. *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. Washington, DC, USA: IEEE Computer Society. pp. 60–67.
- [8] BOXILL, S. A. and YU, L., 2000. An Evaluation of Traffic Simulation Models for Supporting ITS Development. Center for Transportation Training and Research, Texas Southern University, October 2000.
- [9] CHAMPION, A. et al., 1998. Behavioral Road Traffic Simulation with ARCHISIM. *SCSC: Proceedings from the Summer Computer Simulation Conference*, Society for Computer Simulation International, pp. 359-364.
- [10] CHAMPION, A. et al., 1999. Traffic generation with the SCANer II simulator: towards a multi-agent architecture. *DSC '99: Proceedings of the first Driving Simulation Conference*, 1999, pp 311-324
- [11] CHANG, G. and JUNCHAYA, T., 1993. Simulating network traffic flows with a massively parallel computing architecture, *WSC '93: Proceedings of the 25th conference on Winter simulation*, 1993, ACM Press pp762-770.
- [12] CICORTAS, A. and SOMOSI, N., 2005. Multi-Agent System Model for Urban Traffic Simulation, *SACI '05: 2nd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, May 12-14, 2005.
- [13] CLARK, J. and DAIGLE, G., 1997. The importance of simulation techniques in ITS research and analysis, *WSC '97: Proceedings of the 29th conference on Winter simulation*, 1997, ACM Press pp1236-1243.
- [14] DONIKIAN, S. and PHANE, 2001. HPTS: A Behaviour Modelling Language For Autonomous Agents, *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, 2001, ACM Press pp401-408.

- [15] DRESNER, K. and STONE, P. Multiagent traffic management: A reservation-based intersection control mechanism. *The Third International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2004. ACM Press pp. 530-537.
- [16] EHLERT, P. and ROTHKRANTZ, L., 2001. A Reactive Driving Agent for Microscopic Traffic Simulation, *ESM '01: Proceedings of the 15th European Simulation Multiconference, 2001*, SCS Publishing house pp943-949
- [17] EROL, K. et al., 1998. Application of Agent Technology to Traffic Simulation. [online] Available from: <http://www.tfhrc.gov/advanc/agent.htm> [Accessed April 2006]
- [18] FERBER, J., 1999. *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*. England: Addison Wesley.
- [19] HELBING, D. and TREIBER, M., 1998. Jams, waves, and clusters. *Science* 282, 2001.
- [20] HIDAS, P., 2005. Modelling Vehicle Interactions in Microscopic Simulation of Merging and Weaving. *Transportation Research Part C: Emerging Technologies*, **10**, pp. 37-62.
- [21] JONG, S., TORBEN-NIELSEN, B., 2005. SB-MASE User Manual. Institute of Knowledge & Agent Technologies (IKAT), Universiteit Maastricht, the Netherlands.
- [22] KELIN, U. et al., 1998. Traffic simulation based on the high level architecture, *WSC '98: Proceedings of the 30th conference on Winter simulation*, 1998, IEEE Computer Society Press pp1095-1104.
- [23] KLEIN, J., 2002. Breve: a 3D Environment for the Simulation of Decentralized Systems and Artificial Life. *Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*. The MIT Press.
- [24] LEMESSI, M., 2001. An SLX-based microsimulation model for a two-lane road section, *WSC '01: Proceedings of the 33rd conference on Winter simulation*, 2001, IEEE Computer Society pp1064-1071.
- [25] LINDSEY, R. and VERHOEF, E., 2002. Congestion Modelling. *Handbook of Transport Modelling*. 1st reprint 2002 edn. Pergamon, pp. 377-397.
- [26] NAGEL, K., 2004. Multi-Agent Traffic Simulations. Presented at ETH Zurich: Institute for Computer Science.
- [27] NAGEL, K., and SCHRECKENBERG, M., 1992. A Cellular Automaton Model for Freeway Traffic. *Journal de Physique I*, **2**, pp 2221-2229.
- [28] NAGEL, K., Draft as of 02 Feb 2004. *Multi-agent Transportation Simulation*. [online] Available from: <http://www.vsp.tu-berlin.de/publications/matsim-book/> [Accessed December 2005]
- [29] ODELL, J. et al., 2000. Extending UML for agents. *Proceedings of the Second International Workshop on Agent-Oriented Information Systems*. iCue Publishing, 2000.
- [30] ODELL, J., 2003. Agent UML: What is It and Why Do I Care. Presented at ER '03: 22nd International Conference on Conceptual Modelling. Chicago, Illinois, 13-16 October 2003. [online] Available from: <http://www.jamesodell.com/publications.html> [Accessed January 2006]

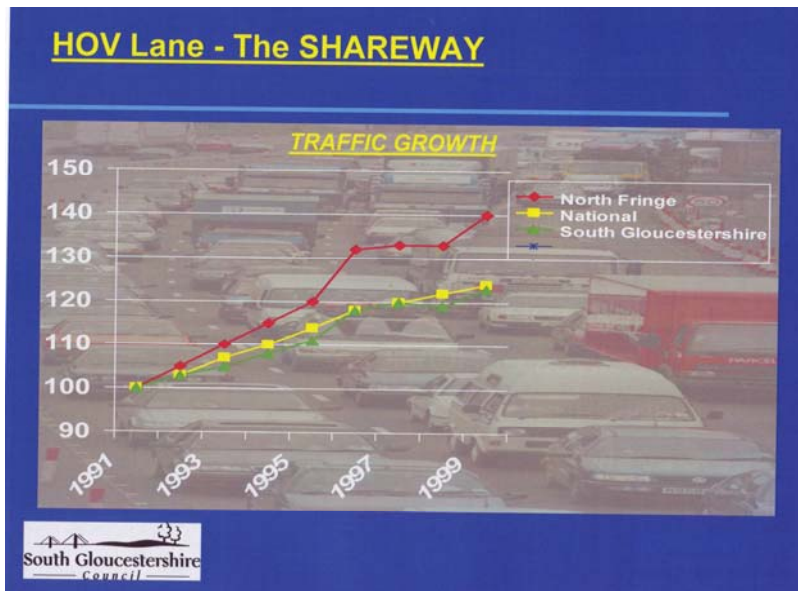
- [31] OWEN, L.E. et al., 2000. Street and traffic simulation: traffic flow simulation using CORSIM, *WSC '00: Proceedings of the 32nd conference on Winter simulation*, 2000, Society for Computer Simulation International pp1143-1147.
- [32] PARUCHURI, P. et al., 2002. Multi agent simulation of unorganized traffic, *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, 2002, ACM Press pp176-183.
- [33] PIVOVAROV, E., 2003. *Traffic Jams* [online]. Available from: <http://physics.ucsd.edu/~epivovar/traffic.htm> [Accessed January 2006]
- [34] PREVEDOUROS, P.D. and WANG, Y., 1999. Simulation of a Large Freeway/Arterial Network with CORSIM, INTEGRATION and WATSim. Paper presented at the 78th Transportation Research Board Annual Meetings, Washington, DC. January 1999.
- [35] RAILSBACK, S.F. et al., Draft as of Jan 2006. Agent-based Simulation Platforms: Review and Development Recommendations.
- [36] RAO, L. et al., 1998. Development and application of a validation framework for traffic simulation models, *WSC '98: Proceedings of the 30th conference on Winter simulation*, 1998, IEEE Computer Society Press pp1079-1086.
- [37] RICHIARDI, M., 2004. The New Italian Road Code and the virtues of the 'shame lane'. Computational Economics 0401002, Economics Working Paper Archive EconWPA.
- [38] RIGOLLI, M. and BRADY, M., 2005. Towards a behavioural traffic monitoring system, *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005, ACM Press pp449-454.
- [39] SALTER, R.J. and HOUNSELL, N.B., 1996. *Highway Traffic Analysis and Design*. Third edn. Palgrave Macmillan.
- [40] SCHULZE, T. and FLIESS, T., 1997. Urban traffic simulation with psycho-physical vehicle-following models, *WSC '97: Proceedings of the 29th conference on Winter simulation*, 1997, ACM Press pp1222-1229.
- [41] SESAM, 2006. *SeSAm: Multi-Agent Simulation Environment* [online]. Available from: <http://www.simsesam.de/> [Accessed April 2006]
- [42] SILVA, V.T.D. et al., 2005. Using the UML 2.0 activity diagram to model agent plans and actions, *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005, ACM Press pp594-600.
- [43] SLINN, M. et al., 1998. *Traffic Engineering: Design, Principles and Practice*. Butterworth-Heinemann.
- [44] SOUTH GLOUCESTERSHIRE COUNCIL, 2006. Personal Communication.
- [45] TANG, W. and WAN, T.R., 2005. Multi-agent Animation Techniques for Traffic Simulation in Urban Environment. *WSCG (Short Papers)*, 2005, pp161-164.
- [46] THE INSTITUTE FOR MODELING COMPLEXITY, 2004. *Understanding the Modeling Process* [online]. Available from: http://netlogo.modelingcomplexity.org/modeling_process.htm [Accessed November 2005]

- [47] TISUE, S. and WILENSKY, U., 2004. NetLogo: Design and Implementation of a Multi-Agent Modeling Environment. Presented at Agent 2004, Chicago, Illinois, October 2004.
- [48] TOMAS, V.R. and GARCIA, L.A., 2005. A cooperative multiagent system for traffic management and control, *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005, ACM Press pp52-59.
- [49] UNITED KINGDOM. Audit Commission. *South Gloucestershire Council – Highways Maintenance Audit*. 2002. London, UK.
- [50] UNITED KINGDOM. Department for Transport. *Perceptions of congestion: report on qualitative research findings*. 2001. London, UK.
- [51] UNITED KINGDOM. Department for Transport. *Transport Ten Year Plan 2000*. 2000. London, UK.
- [52] UNITED KINGDOM. Department for Transport. *Transport Ten Year Plan 2000: Government Response to Select Committee*. 2002. London, UK. (CM 5569)
- [53] UNITED KINGDOM. Government Office for London. *Road Charging Options for London*. 2000. London, UK.
- [54] UNITED KINGDOM. Highways Agency. *High Occupancy Vehicle Lanes Feasibility Study*. 2004. Surrey, UK: KBR.
- [55] UNITED KINGDOM. *UK Government Sustainable Development* [online]
Available from: <http://www.sustainable-development.gov.uk/> [Accessed December 2005]
- [56] WOOLDRIDGE, M., 2002. *An Introduction to MultiAgent Systems*. England, John Wiley and Sons Ltd.

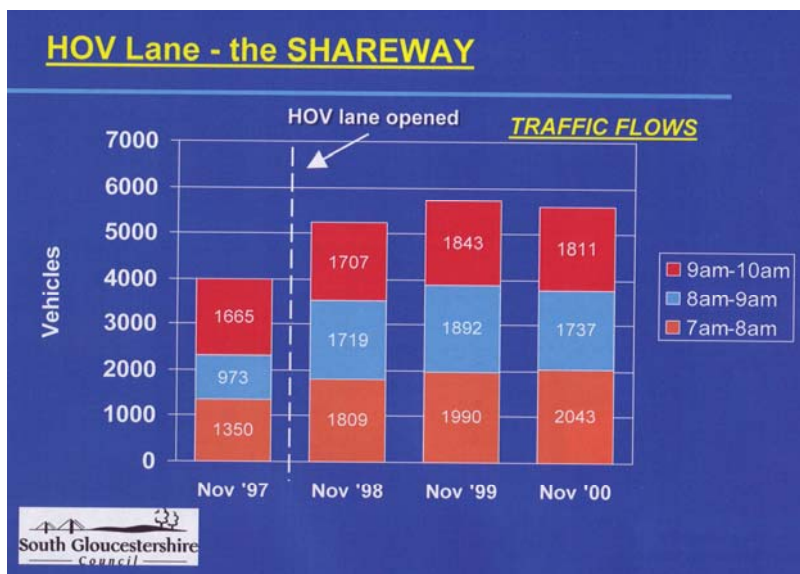
Appendix A: South Gloucestershire HOV Lanes

South Gloucestershire Council kindly provided many details and statistics of the HOV project on the Avon Ring Road. The data as received that has not already been shown is presented in this appendix.

A-1 Statistical Data Received

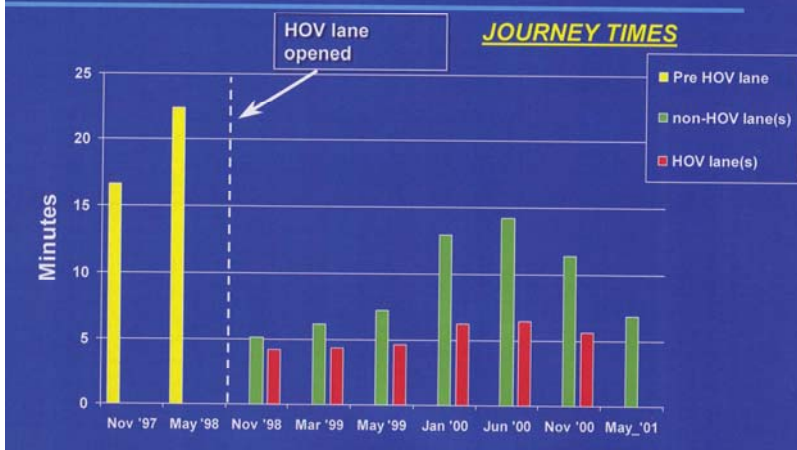


Comparison of national traffic growth with that of South Gloucestershire and the North Fringe (where the Avon Ring Road is situated).



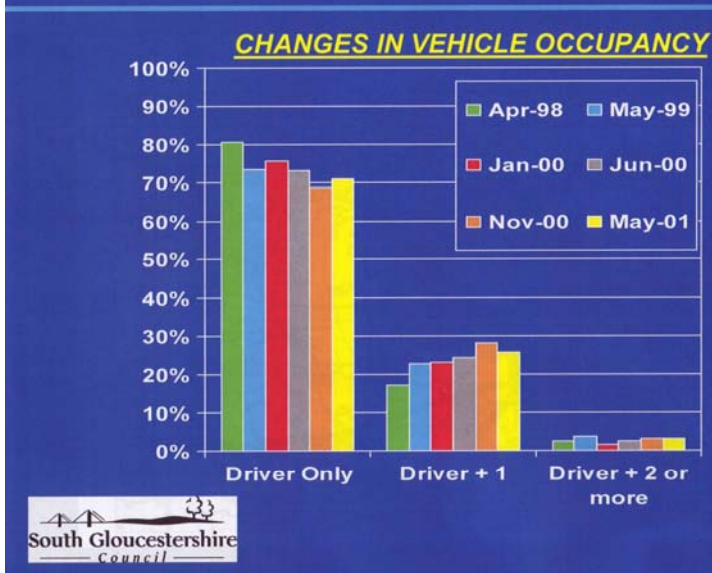
Effects of HOV lane on Traffic Flows.

HOV Lane - the SHAREWAY



Effects of HOV lane on Journey Times.

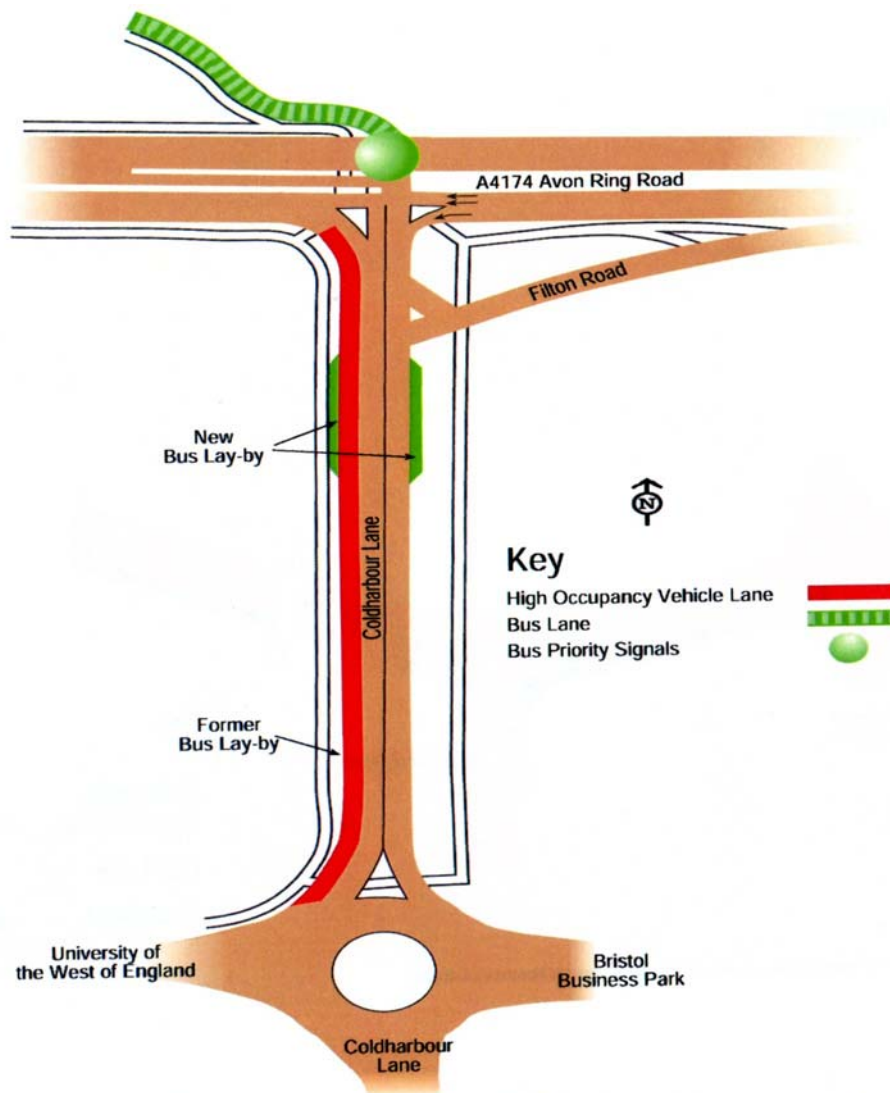
HOV Lane - The SHAREWAY



Effects of HOV lane on Vehicle Occupancy.

A-2 Map Data Received

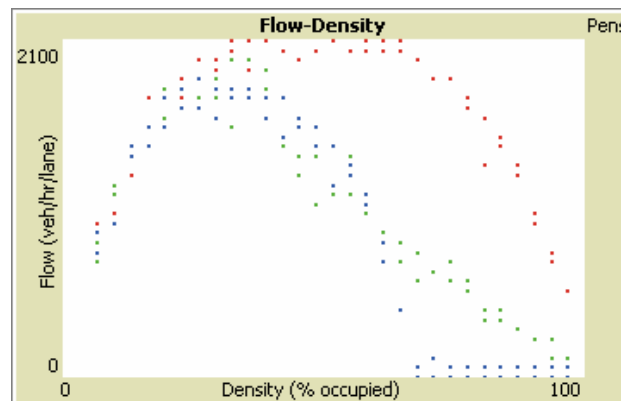
The maps covering the major part of the HOV scheme are displayed in the report. The following map shows the HOV lane on Coldharbour Lane, next to UWE [44]. Due to the short length, complex bus routing, and distance from the other HOV lanes, it was decided not to model this section. A future wider-area study of the traffic network would be able to model this section.



Appendix B: Additional Implementation Details

B-1 Calibrating the Vehicle-following Model

During calibration of the vehicle-following model, altering the behaviour slightly produced major differences in the flow-density curve. The results from three such models are shown below. The green points are from the final model, the red and blue points are from variations on that model that were introduced to improve performance but were rejected due to their effect on vehicle behaviour.



- Green: cars-on patches in-cone (stopping-distance + 1)
- Red: cars in-cone (stopping-distance + 1)
- Blue: cars in-cone (stopping-distance + 1.5)

Above is the NetLogo code used to calculate whether there are any cars within the vehicle's stopping distance (plus one vehicle-length since agents are measured to their center). The NetLogo primitive *in-cone* reports the set of agents in the vehicle's cone of vision. The reason that the green produces more realistic behaviour is that it allows for driver inaccuracies. Red and blue methods determine whether a car is in front by the *exact* distance specified. This turns out to produce some rather predictable and unrealistic behaviour, because it is as if each car uses a precise sensor when determining its distance to the car ahead. This could be true of cars of the future, and it allows vehicles to travel at high speed and close distances, allowing a much higher flow rate at high density akin to the synchronised traffic state.

Blue was an attempt to counter the unrealistic flow rate by looking an extra half-vehicle-length ahead. This worked to an extent, but still produced predictable flows and meant that vehicles would leave half an extra patch between each other when stopped in a jam. The way that the final model works is that first it identifies all the patches within the exact stopping distance, then it finds the set

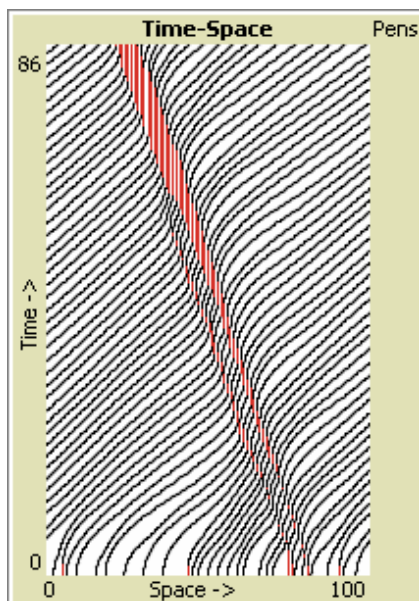
of vehicles on those patches. Because vehicles-on patches only matches vehicles whose *centre* is over the patch, this introduces an up-to one patch error, equivalent to 7.5m. This error appears to produce much more valid vehicle behaviour than the exact measurements.

B-2 Calibrating Vehicle Parameters

The following vehicle parameters were set stochastically by generating a random number between two limits. By making all vehicles behave differently, more realistic and interesting output is gained. These values were chosen as approximate maximum and minimum values taken from a variety of sources.

<i>Parameter</i>	<i>Minimum Value</i>	<i>Maximum Value</i>
Max Speed	20m/s (45mph)	40m/s (90mph)
Max Acceleration	2m/s ² (0-60 in 13s)	4m/s ² (0-60 in 7s)
Max Deceleration	5m/s ²	8m/s ²

B-3 Example Time-Space Charts



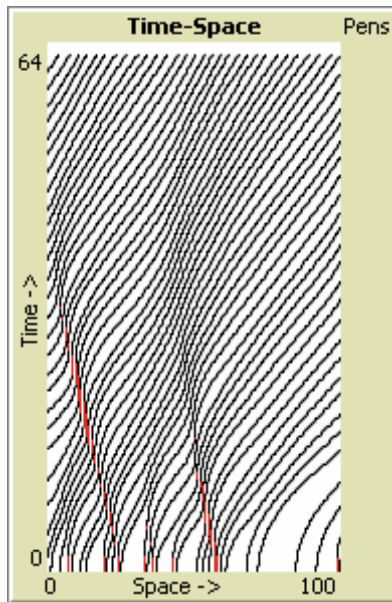
Road: One lane, 100 patches long (7.5km)

Vehicles: 30 vehicles (30% density)

Time Slice: 0.2

Explanation:

This shows two queues building up and then merging into a single queue.



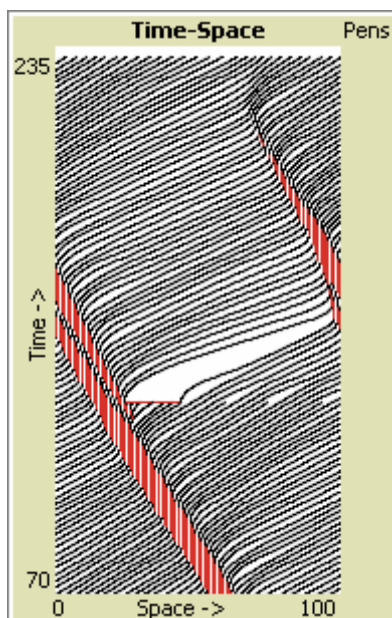
Road: One lane, 100 patches long (7.5km)

Vehicles: 30 vehicles (30% density)

Time Slice: 0.2

Explanation:

This shows queues dispersing and the traffic slowly synchronising.



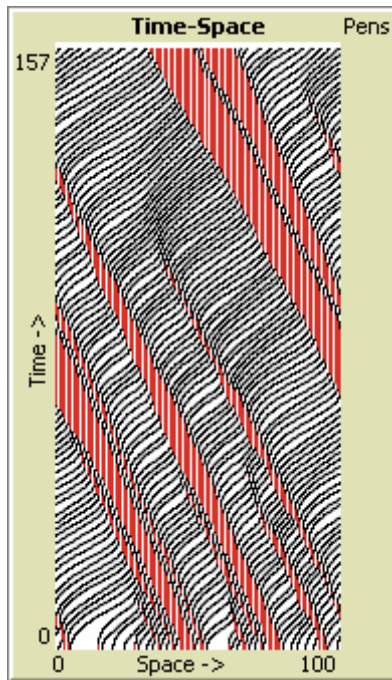
Road: One lane, 100 patches long (7.5km)

Vehicles: 35-27 vehicles

Time Slice: 0.2

Explanation:

This started with 35 vehicles, which forms a standard queue. Then 8 vehicles were removed, leaving 27 remaining. This shows how traffic can revert to free-flow when the density falls below critical level.



Road: One lane, 100 patches long (7.5km)

Vehicles: 50 vehicles

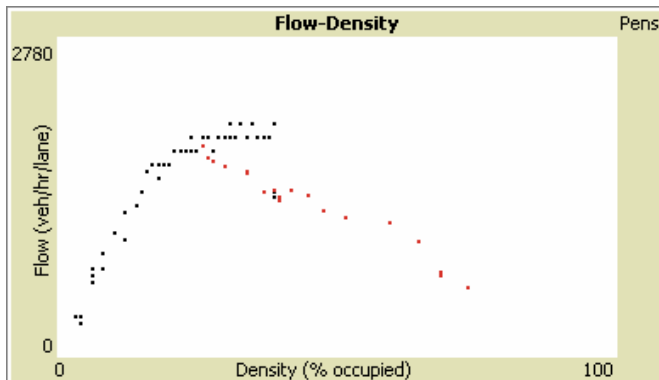
Time Slice: 0.2

Explanation:

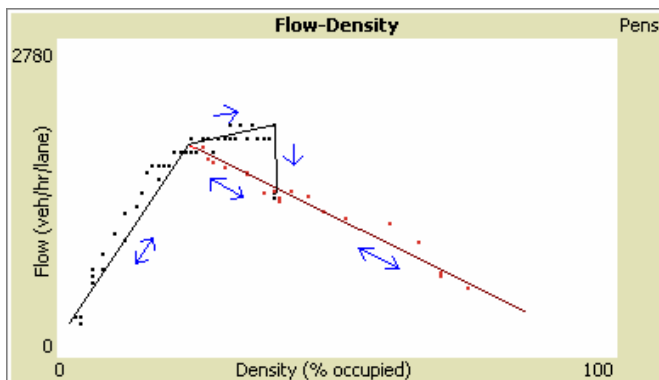
This shows how 50% density can result in one long two-part jam from many short ones.

B-4 Example Flow-Density Chart

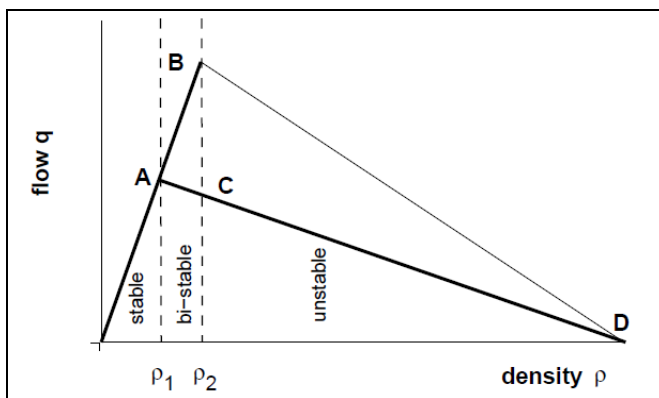
This example shows the state change from stable to bi-stable to unstable and how the bi-stable state is only possible from the stable state. The road is 100 patches long, and begins with 5 vehicles. The number of vehicles is increased by adding a new vehicle in the largest gap, and giving it the same speed as the vehicle in front of it.



Black points show those readings from the start of the simulation until the critical density was reached. Red points are readings after this point, i.e. during congestion.



It is easy to see the point at which the critical density is reached; the vehicle flow drops dramatically from 2000 vph to 1470 vph at 39% density (52 v/km). The effect on the traffic was the flow changing from free-flowing to a large, moving queue. Once in this state, free-flow cannot return until the density lowers to 26% (35 v/km).



The one-way hysteresis effect can be depicted with arrows and clearly resembles the fundamental diagram. The only difference is that once the density reaches critical, the flow immediately drops to the congested flow rate, with no B-D state as suggested by the flow diagram.

The reason that the other flow-density charts do not show this phenomenon is that they were created by resetting the vehicles between each density run rather than altering the density at runtime. This allowed the maximum flow possible to be shown. In fact, if the two types of diagram are combined, this shows the possible emergence of the B-D state.

Appendix C: Simulation Code

This is the contents of the procedures tab in NetLogo for the simulation model. Many of the procedures are called by buttons on the interface, and the rest are called by those procedures.

```

;;;;;;;;;;;;;
;;;;;;;;; EDITING TOOLS
;;;;;;;;;;;;;

; Global procedure - Apply effects of selected editing tool.
to edit-world

  if mouse-down? [
    ; Store patch under cursor in local variable
    let mouse-patch patch-at mouse-xcor mouse-ycor

    ; Perform action on patch depending on edit tool
    if EDIT_TOOL = "Road" or EDIT_TOOL = "HOV Road" [
      ask mouse-patch [
        set road? true

        ifelse EDIT_TOOL = "HOV Road" [
          set hov? true
          set pcolor red - 2
        ] [
          set hov? false
          set pcolor black
        ]
      ]
      set roads patches with [road?] ; re-evaluate roads set
    ]

    if EDIT_TOOL = "Grass" [
      ask mouse-patch [
        set pcolor green - 2.5
        set road? false
        set hov? false
        set directions []
        set plabel ""
      ]
      set roads patches with [road?] ; re-evaluate roads set
    ]

    ; Sprout or remove flow counter on clicked patch.
    if EDIT_TOOL = "Flow Counter" [
      ifelse any? counters-on mouse-patch
        [ask counters-on mouse-patch [die]]
        [ask mouse-patch [sprout-counters 1 [init-counter]]]
      wait 0.1
    ]

    ; Sprout or remove vehicle on clicked patch.
    if EDIT_TOOL = "Car" [
      ifelse any? vehicles-on mouse-patch
        [ask vehicles-on mouse-patch [die]]
        [ask mouse-patch [sprout-vehicles 1 [init-vehicle]]]
      wait 0.1
    ]
  ]

```

```

; Sprout or remove traffic lights on clicked patch.
if EDIT_TOOL = "Traffic Lights" [
  ifelse any? lights-on mouse-patch
  [ask lights-on mouse-patch [die]]
  [
    ; Try reading ints from user; throw error if cannot convert (sum converts to number)
    carefully [
      let EW sum (list read-from-string user-input "How many seconds to stay green on
horizontal link?")
      let NS sum (list read-from-string user-input "How many seconds to stay green on vertical
link?")
      ask mouse-patch [
        sprout-lights 1 [
          init-lights EW NS      ; Set the green times horiz and vertical
        ]
      ]
      user-message "You didn't enter a number; try again."
    ]
    wait 0.1
  ]

; Sprout or remove source on clicked patch.
if EDIT_TOOL = "Source" [
  ifelse any? sources-on mouse-patch
  [ask sources-on mouse-patch [die]]
  [
    ; Try reading ints from user; throw error if cannot convert (sum converts to number)
    carefully [
      let inp sum (list read-from-string user-input "Sprout a vehicle every how many
seconds?")
      ask mouse-patch [
        sprout-sources 1 [
          set influx-every inp      ; This source will create a car every inp secs
          set label influx-every ; Show how often you create a car.
        ]
      ]
      user-message "You didn't enter a number; try again."
    ]
    wait 0.1
  ]

; Sprout or remove sink from selected patch.
if EDIT_TOOL = "Sink" [
  ifelse any? sinks-on mouse-patch
  [ask sinks-on mouse-patch [die]]
  [ask mouse-patch [sprout-sinks 1 []]]
  wait 0.1
]
]
end

; Global procedure. Applies selected speed limit to selected road.
to edit-speed-limit

; Always show speed limits
ask roads [set plabel int (speed-limit * 3.6)]      ; Convert m/s to km/h

if mouse-down? [
  ; Store patch under cursor in local variable
  let mouse-patch patch-at mouse-xcor mouse-ycor

  ; Set speed limit in meters per second.
  set speed-limit-of mouse-patch SPEED_LIMIT * 1000 / (60 * 60)
]
end

```

```

; Global procedure. Applies selected direction to selected road.
to edit-road-directions

  ; Always show directions
  ask roads [set plabel list-to-string directions]

  if mouse-down? [
    ; Store patch under cursor in local variable
    let mouse-patch patch-at mouse-xcor mouse-ycor

    ; If clicking on a road, set it's directions.
    if road?-of mouse-patch [
      ; Apply chosen direction to selected road.
      if ROAD_DIRECTION = "North" [ask mouse-patch [edit-road-directions-list "N"]]
      if ROAD_DIRECTION = "East"  [ask mouse-patch [edit-road-directions-list "E"]]
      if ROAD_DIRECTION = "South" [ask mouse-patch [edit-road-directions-list "S"]]
      if ROAD_DIRECTION = "West"  [ask mouse-patch [edit-road-directions-list "W"]]
      if ROAD_DIRECTION = "Clear" [ask mouse-patch [set directions []]]
    ]
  ]
end

; Road procedure. Applies given direction char (N/E/S/W) to this road.
to edit-road-directions-list [direct]

  ; Add this direction to existing ones.
  set directions sort fput direct directions ; union existing list

  ; This used to set a whole line of roads at once. It didn't work great so was removed.
end

; Global procedure. Loads selected world into the simulator.
to load-model
  let filename user-choose-file
  if (filename != false) [
    import-world filename
  ]
end

; Global procedure. Saves model state to selected file.
to save-model
  let filename user-choose-new-file
  if (filename != false) [
    export-world filename
  ]
end

; Global procedure. Sets model state to default. Same as reloading NetLogo.
to clear-model
  set-default-shape vehicles "car top"
  set-default-shape sources "circle"
  set-default-shape sinks "target"
  set-default-shape counters "flow counter"
  set-default-shape lights "Traffic Light"

  set PATCH_LENGTH 7.5 ; meters
  ask turtles [die]

  ; Set all patches to 'grass'
  ask patches [
    set directions []
    set pcolor green - 2.5
    set plabel ""
    set speed-limit 0 ; no limit
    set road? false
    set hov? false
  ]

  set roads patches with [road?] ; Re-evaluate road set (will be empty)
  setup ; Set all per-run variables to defaults.
end

```

```

;;;;;;;;;;
;;; VARIABLES
;;;;;;;;;;

; Each of these is a type of agent, and has its own set of parameters.
breed [vehicles vehicle]
breed [lights light]
breed [sources source]
breed [sinks sink]
breed [counters counter]

; Global variables.
globals [
  PATCH_LENGTH ; usually 7.5m as per standard CA traffic simulation.
  SIMULATED_TIME ; Current simulation time elapsed in seconds. Speed depends on time slice.
  REAL_TIME_DIFF ; Number of simulation seconds per realtime second. Used for real-time switch.
  REAL_TIME_RATIO ; To show speed ratio

  FLOW_COUNTER ; Total number of vehicles to cross flow points.
  LAST_PLOT_TIME ; Simulated time of last plot. So we dont plot many times per simulated second.
  ALL_LANE_CHANGE ; Sim time of last global lane change, do it once a simulated second
  roads ; Agentset of all road patches. equivalent to "patches with [road?]"

  VEHICLES_EXITED ; For AVO calculation.
  PEOPLE_EXITED ; For AVO calculation.
]

; Patch can be road or grass. If a road, it can be HOV or not, and has directions and speed limit.
patches-own [
  road? ; Is this patch a road?

  directions ; List of strings N/E/S/W - which direction(s) can be taken from this patch?
  speed-limit ; in m/s, 0 = no limit.
  hov? ; Boolean, true = only cars with 2+ people can use the road.
]

vehicles-own [
  LAST_PLOT_X ; So we can detect when plot wraps.
  LAST_TURN_PATCH ; So we only evaluate turning once per patch.
  LAST_LANE_CHANGE ; So we dont change lanes too often.
  TIME_ENTERED ; to calculate journey time.

  speed ; Current speed. In meters per second. (m/s) Patch = 7.5m
  desired-speed ; Driver's desired speed m/s (always try to reach and maintain this)
  max-speed ; Vehicles maximum speed m/s
  max-acceleration ; Max difference in speed per tick when accelerating m/s2
  max-deceleration ; Max difference in speed per tick when decelerating m/s2

  passengers ; Number of people in the vehicle.
]

lights-own [
  green-time-h ; How many simulated seconds to stay green for horizontal direction.
  green-time-v ; same for vertical
  yellow-time ; How many simulated seconds to stay yellow after green.
  event-time ; What simulation time does the light next need to change state?
]

sources-own [
  influx-every ; How often in simulated seconds to create new vehicles
  last-influx ; Simulated time of last vehicle creation
]

counters-own [
  last-vehicle ; The latest vehicle to go over this counter for flow calculations.
]

```

```

;;;;;;;;;;;;;
;;;;;;;;;;;;; INITIALIALISATION
;;;;;;;;;;;;;

; Global procedure. This runs when the simulation is loaded.
to startup
  clear-model      ; Set model state to default.

  ; Automatically clear road labels (i.e speed limit/directions) when tool turned off.
  loop [
    every 1 [
      ask roads [set plabel ""]
    ]
  ]
end

;;;;;;;;;;;;;
;;;;;;;;;;;;; SETUP PROCEDURES
;;;;;;;;;;;;;

; Global proc. Do everything required to reset a simulation run.
to setup
  ask lights [set event-time event-time - int SIMULATED_TIME] ; stay in current sync
  set SIMULATED_TIME 0
  set LAST_PLOT_TIME 0 ; So it doesn't plot flow 0 after reset
  set FLOW_COUNTER 0
  set REAL_TIME_DIFF -1 ; HACK to stop the flow plotter from crashing when div by 0
  set REAL_TIME_RATIO 1 ; No way to know how fast it will run. Guess in real time.
  set VEHICLES_EXITED 0
  set PEOPLE_EXITED 0

  ask vehicles [set LAST_LANE_CHANGE 0] ; No car has ever changed lanes

  ; Clear and reset plots
  set-current-plot "Journey Times" clear-plot
  set-current-plot "AVO" clear-plot
  init-timespace ; Clear and Reset time-space plot
end

; Global proc. Kill all vehicles, then make new ones on random roads.
to regenerate-vehicles
  if count roads < NUMBER_OF_VEHICLES [
    user-message "There are not enough roads for that number of vehicles. Setup will halt."
    stop
  ]

  ask vehicles [die]

  repeat NUMBER_OF_VEHICLES [
    ask one-of roads with [count vehicles-here = 0] [
      sprout-vehicles 1 [
        init-vehicle
      ]
    ]
  ]
end

; Vehicle proc. Run when every new vehicle agent is born
to init-vehicle
  let road-heading "E"
  if not empty? directions [set road-heading one-of directions];pick a direction in which road goes.

  ; Default is East incase of no directions (eg blank world)
  set heading heading-from-direction road-heading

  set speed 0 ; All cars start "stopped"
  set max-speed (20 + random 20) ; 60 mph = 26.8 m/s. 100 mph = 44.7 m/s.
  set max-acceleration 2 + random 2 ; Slow cars 0-60 in 13secs (2 m/s²), fast cars 7secs (4 m/s²)
  set max-deceleration 5 + random 3 ; Braking power 5 m/s² to 8 m/s².

  ifelse random 100 < %_HOV [ ; Make 2+ cars with %_HOV probability
    set passengers 2 + random 4 ; Cars have 2 to 5 passengers. Try higher to make up for buses?
    set color red
  ] [
    set passengers 1
    set color blue
  ]
end

```

```

set LAST_PLOT_X 0
set LAST_TURN_PATCH nobody
set TIME_ENTERED SIMULATED_TIME ; Make a note of what time I entered, for journey times.
end

; Lights proc. Run when each traffic light agent is born.
to init-lights [h v]
set heading 90 ; Greens start on North-south link (V)
set color green ; Green or yellow
set green-time-h h ; green time for east-west link
set green-time-v v ; and for north-south link.
set yellow-time 3
set event-time int SIMULATED_TIME + green-time-v ; Queue first event.
end

; Counter proc. Sets flow counter to face right direction, visual only.
to init-counter
if not empty? directions [
set heading heading-from-direction one-of directions
]
end

; Global proc. Initialise time-space graph.
to init-timespace
set-current-plot "Time-Space"
clear-plot
set-plot-x-range min-pxcor - 0.5 max-pxcor + 0.5 ; So autoplot doesnt adjust when points go out
end

;;;;;;;;;;;;;
;;;;;;;;;;;;; RUNTIME PROCS
;;;;;;;;;;;;;

; Global proc. Runs one time-step of simulation. I believe the order of calls is important.
to go
no-display ; Don't update visualisation until all objects updated.

if PLOT_TIMESPACE? [ ; Save some cycles if we dont want them
measure-timespace ; update the time-space data and graph
]

measure-flow ; update flow-density data and graph
update-lights ; Change traffic lights when they time out

if SOURCES_ACTIVE? [
process-sources ; Add cars if required from sources
]

update-vehicles ; move all vehicles
process-sinks ; Remove cars which are on a sink
update-time ; must be last (or first i suppose)

display ; Update the visualisation now.
end

; Global proc. Update time-space graph.
to measure-timespace
set-current-plot "Time-Space"

ask vehicles [
create-temporary-plot-pen (word who) ; Creates if it doesnt exist, uses if it does

; Cars in a queue show up red!
ifelse speed = 0 [set-plot-pen-color red] [set-plot-pen-color black]

; This means no horizontal lines when plotting from xmax to xmin. Only for EASTBOUND cars!
ifelse LAST_PLOT_X > xcor [
plot-pen-up
plotxy xcor SIMULATED_TIME
plot-pen-down
] [
plotxy xcor SIMULATED_TIME
]
set LAST_PLOT_X xcor
]
end

```

```

; Global sinks proc. Ask sinks to remove any vehicles on them
to process-sinks
  ask sinks [
    ask vehicles-here [
      ; Draw on the journey times chart
      set-current-plot "Journey Times"

      ; Plot this vehicle on journey times graph.
      ifelse passengers > 1 [set-current-plot-pen "hov"] [set-current-plot-pen "car"]
      plotxy SIMULATED_TIME SIMULATED_TIME - TIME_ENTERED

      die ; Goodbye world.
    ]
  ]
end

; Global sources proc. Ask sources to make new vehicle as often as they are set to.
to process-sources
  ask sources [

    ; If it's time for a new vehicle to be created, and one hasn't already been created.
    if (int SIMULATED_TIME != last-influx) and (int SIMULATED_TIME mod influx-every = 0) [
      set last-influx int SIMULATED_TIME

      ; Dont create new vehicles on top of queuing ones
      if not any? vehicles-here [
        ask patch-here [
          sprout-vehicles 1 [
            init-vehicle

            ; Gets the vehicle in front or nobody.
            let vehicle-infront min-one-of vehicles with [self != myself] in-cone 20 0 [distance
myself]
            ; If a vehicle was infront, copy its speed.
            ifelse vehicle-infront != nobody [
              set speed speed-of vehicle-infront
            ] [
              set speed min list max-speed speed-limit
            ]
          ]
        ]
      ]
    ]
  ]
end

; Global lights proc. Change the lights if the event time is reached.
to update-lights

  ; Show seconds until event as lights label.
  every 1 [ask lights [set label event-time - int SIMULATED_TIME]]

  ask lights [
    ; A light change is scheduled...
    if int SIMULATED_TIME = event-time [
      ifelse color = yellow [ ; Going from yellow/red to red/green
        set color green
        rt 90

        ; Heading defines which direction the green light faces. Set next event.
        ifelse heading = 0 or heading = 180 [
          set event-time int SIMULATED_TIME + green-time-h
        ] [
          set event-time int SIMULATED_TIME + green-time-v
        ]
      ]
      ; need to go from red/green to red/yellow.
      [
        set color yellow
        set event-time int SIMULATED_TIME + yellow-time
      ]
    ]
  ]
end

```

```

; Global counter proc. Update counters and flow/density and AVO graphs.
to measure-flow
  if any? counters [
    ; Ask all flow counters to check if a new vehicle is on them. Doesn't work with 1 vehicle.
    ask counters [
      if any? vehicles-here with [who != last-vehicle-of myself] [
        set last-vehicle who-of one-of vehicles-here ; Flow stuff
        set FLOW_COUNTER FLOW_COUNTER + 1

        set VEHICLES_EXITED VEHICLES_EXITED + 1 ; AVO stuff
        set PEOPLE_EXITED PEOPLE_EXITED + passengers-of one-of vehicles-here
      ]
    ]

    ; Plot flow and density, and AVO once per simulated minute.
    if (int SIMULATED_TIME != LAST_PLOT_TIME) and (int SIMULATED_TIME mod FLOW_PLOT_PERIOD = 0) [

      ; FLOW STUFF
      set LAST_PLOT_TIME int SIMULATED_TIME ; So we don't plot again this second.

      let density (count vehicles / count roads * 100)
      let flow FLOW_COUNTER * 60 * (60 / FLOW_PLOT_PERIOD) / count counters
      set FLOW_COUNTER 0

      set-current-plot "Flow-Density"
      plotxy density flow

      ; AVO STUFF
      if VEHICLES_EXITED > 0 [
        set-current-plot "AVO"
        plotxy SIMULATED_TIME PEOPLE_EXITED / VEHICLES_EXITED
        set PEOPLE_EXITED 0
        set VEHICLES_EXITED 0
      ]
    ]
  ]
end

; Global proc. Update the position and variables for all vehicles.
to update-vehicles

  ; Processes changes of direction. Before lane change else vehicle looks at directions wrong.
  ask vehicles [
    vehicle-turn-behaviour
  ]

  ; Lane changing and turning is completed before any vehicles decide speed or move.
  ; Check all vehicles for lane change worthiness every 1 sim second (ALL_LANE_CHANGE time).
  if int SIMULATED_TIME != ALL_LANE_CHANGE [
    set ALL_LANE_CHANGE int SIMULATED_TIME ; So we don't change lanes again this second.
    ask vehicles [
      if SIMULATED_TIME > LAST_LANE_CHANGE + 5 [ ; Dont change lanes except every 5 seconds
        without-interruption [lane-changing] ; So other vehicles don't see you moving.
      ]
    ]
  ]

  ; Vehicles decide their desired speed based on the environment.
  ask vehicles [
    ; Run the vehicle following scheme. This sets the driver's desired speed.
    simple-vehicle-following

    ; Vehicle-following scheme can be overridden by traffic light behaviour.
    traffic-light-behaviour
  ]

  ; Ensure all vehicles have decided before any vehicle moves position.
  ask vehicles [
    move-vehicle ; Update vehicle speed and position based on desired-speed
  ]
end

```

```

; Vehicle proc. Updates the speed and position of the vehicle.
to move-vehicle
; Change vehicle speed if it is not equal to max "possible" speed
if speed != min list desired-speed max-speed [

; Need to go faster = first block
ifelse (speed < desired-speed)
[
set speed speed + (max-acceleration * TIME_SLICE)
if speed > desired-speed [set speed desired-speed] ; Restrict to desired speed
if speed > max-speed [set speed max-speed] ; Restrict by maximum vehicle speed.
]
[ ; need to slow down
set speed speed - (max-deceleration * TIME_SLICE)
if speed < desired-speed [set speed desired-speed]
]
]

; Move the amount you would move in a single time-slice, in patches not meters.
fd speed * TIME_SLICE / PATCH_LENGTH
end

; Vehicle proc. Checks for mandatory and optional turning points in road.
to vehicle-turn-behaviour

; If this road has directions set, and it hasn't been evaluated yet for this vehicle...
if not empty? directions and patch-here != LAST_TURN_PATCH [
let my-dir direction-from-heading heading ; Get vehicle heading char (NESW)

; If the road has turned, pick one of the new directions at random.
ifelse not member? my-dir directions [
set heading heading-from-direction one-of directions
setxy pxcor pycor ; inaccurate. should go back however much were onto patch.

; hack because we can't see round corners. If a car is now ahead, slow to match its speed.
let new-leader min-one-of vehicles with [self != myself] in-cone (stopping-distance + 1) 0
[distance myself]
if new-leader != nobody [
set speed speed-of new-leader
]
] [ ; My current direction is still valid

; If the road has other directions, change direction with a 1 in 5 chance.
if length directions > 1 and random 5 = 0 [

; Pick a new direction to think about
let old-heading heading
set heading heading-from-direction one-of remove my-dir directions

; Check if you're in the right lane to make this turn:
; Does next patch go in my old direction? If so, wrong lane!
ifelse member? my-dir directions-of patch-ahead 1 [
set heading old-heading
] [
; Move to center of patch to check for vehicles in-cone
let oldx xcor
let oldy ycor
setxy pxcor pycor

; If theres a car too close to stop for, dont turn. i.e. change back heading and position.
ifelse any? vehicles in-cone (stopping-distance + 1) 5 with [self != myself] [
set heading old-heading
setxy oldx oldy
] [
; TURNED! Treat as a lane change - dont change lanes straight after turn.
set LAST_LANE_CHANGE SIMULATED_TIME
]
] ; right lane?
] ; other directions?
] ; if not forced to turn

; Remember the patch that we just evaluated so we don't do it again just yet.
set LAST_TURN_PATCH patch-here
]
end

```

```

; Vehicle proc. Think about changing lanes and do it if you like.
to lane-changing

  ; Only think about changing lanes if the current road only has one direction
  if length directions = 1 [

    ; Remember the current vehicle ahead of you, if they are imminent to slowing you down
    let old-leader min-one-of vehicles with [self != myself] in-cone (stopping-distance + 2) 0
    [distance myself]

    ; INCENTIVE: Boolean whether or not current lane is free i.e. no slower cars close ahead
    let straight-free old-leader = nobody or (speed-of old-leader > speed)

    ; Incentive: If you're on the HOV but you don't have 2+. could factor 90% dont break rule.
    let must-change (hov? and passengers < 2)

    ; If no leader, or the leader isnt slowing you down, think about moving left.
    if straight-free [
      lane-change-left
    ]

    ; Your leader is slowing you down or you are in HOV illegally. Consider moving right.
    if not straight-free or must-change [
      lane-change-right
    ]
  ]
end

; Vehicle proc. Check safety to move to the left.
to lane-change-left

  ; If there is a lane to the left
  if directions = directions-of patch-left-and-ahead 90 1 [

    ; Move to it, face backwards so you can look over where u would be placed and check safety
    lt 90 jump 1 lt 90 jump -1

    ; any cars under safe distance between cars, Assuming we're going faster than them.
    let new-follower min-one-of vehicles with [self != myself] in-cone 3 0 [distance myself]

    ; If no vehicle behind, back safety OK.
    ifelse new-follower = nobody and (not hov? or passengers > 1) [

      jump 1 lt 180 ; face forward

      ; Get nearest vehicle in front on new lane, if they are within stopping distance.
      let new-leader min-one-of vehicles with [self != myself] in-cone (stopping-distance + 1) 0
      [distance myself]

      ; Finally, if the new leader (within stopping distance) is going slower than you then ABORT.
      ifelse new-leader != nobody and speed-of new-leader < speed [
        rt 90 jump 1 lt 90 ; move back
      ] [
        set LAST_LANE_CHANGE SIMULATED_TIME ; Lane change complete!
      ]

    ] [
      ; not safe because of vehicles behind.
      jump 1 lt 90 jump 1 lt 90 ; move back
    ]
  ]
end

```

```

; Vehicle proc. Change lanes to the right if it is safe.
to lane-change-right

  ; If there is a lane to the right - logistics.
  if directions = directions-of patch-right-and-ahead 90 1 [

    ; Move to it, face backwards so you can look over where u would be placed
    rt 90 jump 1 rt 90 jump -1

    ; Follower, if potentially too close to stop. 10 = max stopping distance, actually ~20 patches.
    let new-follower min-one-of vehicles with [self != myself and stopping-distance + 2 > distance
myself] in-cone 10 0 [distance myself]

    ; If no vehicle close behind, or they are going slower than you, consider moving right.
    ifelse new-follower = nobody or (speed-of new-follower < speed and distance new-follower > 2) [

      jump 1 lt 180          ; face forward

      ; Closest car in front within stopping distance, on new lane.
      let new-leader min-one-of vehicles with [self != myself] in-cone (stopping-distance + 1) 0
[distance myself]

      ; Finally, if the new leader (within stopping distance) is going slower than you then ABORT.
      ifelse new-leader != nobody and (speed-of new-leader < speed) [
        lt 90 jump 1 rt 90          ; move back
      ] [
        set LAST_LANE_CHANGE SIMULATED_TIME ; Lane change complete!
      ]

    ] [
      jump 1 rt 90 jump 1 rt 90    ; Not safe because of vehicles behind.
      ; move back
    ]
  ]
end

; Vehicle proc. car following rule - If vehicles within stopping distance, slow down. Else speed up.
to simple-vehicle-following

  ; Agentset of all vehicles within my stopping distance in front (+ 1 car's space)
  let vehicles-infront (vehicles-on patches in-cone (stopping-distance + 1) 0)

  ; Apply vehicle following rule to desired-speed, applying speed limit.
  ifelse any? vehicles-infront with [self != myself] ; Any cars infront? excluding myself
  [set desired-speed 0]
  [ifelse speed-limit = 0 [set desired-speed max-speed] [set desired-speed speed-limit]]
end

; Vehicle proc. Obey traffic lights.
to traffic-light-behaviour

  ; Ignore the lights if you are already on them (or adjacent ones)
  if not any? lights-here [

    ; All lights within stopping distance. Ignore those you're going too fast to stop for.
    let lights-infront lights in-cone (stopping-distance + 1) 0 with [distance myself > value-from
myself [stopping-distance]]

    ; If any lights infront which are red in my direction, stop!
    ifelse any? lights-infront with [member? subtract-headings heading heading-of myself list 0 180]
    [set desired-speed 0]

    ; Else could still be yellow/green lights infront. if yellow stop too.
    [if any? lights-infront with [color = yellow] [set desired-speed 0]]
  ]
end

; Vehicle reporter (proc that returns). Current Stopping distance in patches.
to-report stopping-distance
  report ((speed ^ 2) / (2 * max-deceleration)) / PATCH_LENGTH
end

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;; SIMULATION TIME MANAGEMENT ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Global proc. Update the current simulation time. Called every tick.
to update-time

  ; Auto-adjust the time slice so the simulation runs in real time. This runs every REAL sec.
  every 1 [
    set REAL_TIME_RATIO SIMULATED_TIME - REAL_TIME_DIFF

    if REAL_TIME? [
      set TIME_SLICE TIME_SLICE / REAL_TIME_RATIO
    ]
    set REAL_TIME_DIFF SIMULATED_TIME
  ]

  ; Update simulation time by the time slice value.
  set SIMULATED_TIME SIMULATED_TIME + TIME_SLICE
end

; Reports a string in the form "X hours X mins X secs" representing current simulation time.
to-report readable-time-string
  let calc-mins int (SIMULATED_TIME / 60)
  let hrs int (calc-mins / 60)
  let secs int (SIMULATED_TIME - (calc-mins * 60))
  let mins int (calc-mins - (hrs * 60))

  report hrs + " hours " + mins + " mins " + secs + " secs"
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;; HELPER FUNCTIONS ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Report the heading that corresponds to the NESW direction (string)
to-report heading-from-direction [dir]
  if dir = "N" [report 0]
  if dir = "E" [report 90]
  if dir = "S" [report 180]
  if dir = "W" [report 270]
  report false
end

; Report the direction (NESW) that corresponds to the heading number.
to-report direction-from-heading [head]
  if head = 0 [report "N"]
  if head = 90 [report "E"]
  if head = 180 [report "S"]
  if head = 270 [report "W"]
  report false
end

; Reports the string made up of concatenating values in the given list.
to-report list-to-string [l]
  let s ""
  foreach l [
    set s (s + ?)
  ]
  report s
end

```

Appendix D: Graphical Interface

There is not enough space on the screen to display the three charts on the right, so the user has to scroll the interface in order to see them. This is a limitation of NetLogo.

